(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number: **0 474 339 A2**

(12) EUROPEAN PATENT APPLICATION

(21) Application number: **91306127.1**

(51) Int. Cl.5: **G06F 9/44**

(22) Date of filing: **05.07.91**

(30) Priority: **14.08.90 US 567303**

(43) Date of publication of application:
11.03.92 Bulletin 92/11

(84) Designated Contracting States:
DE FR GB IT NL

(71) Applicant: **DIGITAL EQUIPMENT CORPORATION**
111 Powdermill Road
Maynard Massachusetts 01754-1418(US)

(72) Inventor: **Travis, Robert L., Jr.**
1547 Main Street
Concord, Massachusetts 01742(US)

Inventor: **Jacobson, Neal F.**
6 Cranwell Court
Nashua, New Hampshire 03062(US)
Inventor: **Wilson, Andrew P.**
Commons Brink, Bunces Lane, Burghfield Common
Reading RG7 3DP(GB)
Inventor: **Renzullo, Michael J.**
7 Byron Road
Ashland, Massachusetts 01721(US)

(74) Representative: **Goodman, Christopher et al**
Eric Potter & Clarkson St. Mary's Court St. Mary's Gate
Nottingham NG1 1LE(GB)

(54) Methods and apparatus for providing a client interface to an object-oriented invocation of an application.

(57) In response to a message requesting a method invocation from an application or user, a client application determines the proper method to be invoked by retrieving information from a class data base, comparing the retrieved information with user preferences, and selecting the proper method based upon the comparison. Server connection and start-up involves locating a platform capable of executing code associated with the selected method and, if necessary, executing a process to start an application associated with the selected method.

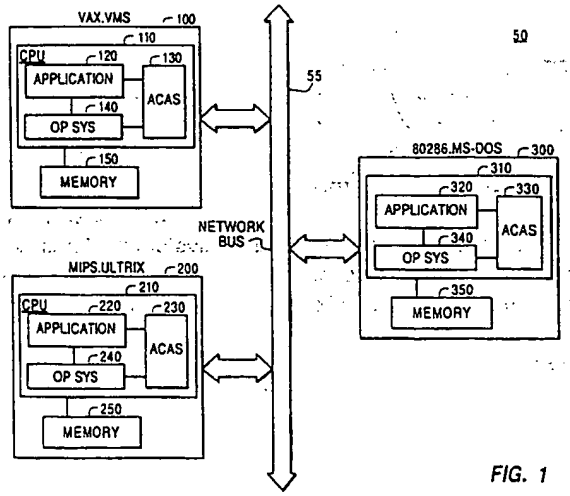

FIG. 1

EP 0 474 339 A2

simple manner for applications on different platforms to communicate with each other, such as through a uniform and consistent interface for applications. There is also a need for a dynamic invocation environment for applications in a distributed heterogeneous environment. This includes providing the ability to invoke applications that are not yet loaded and running as well as those which are.

III. SUMMARY OF THE INVENTION

To achieve these needs, the present invention provides for interaction of processes in an object-oriented manner by which a system manages "classes" of data instances and applications rather than managing the data itself. The management of such classes can involve a data base which contains information about the classes, such as certain common attributes of applications or instances which are supported by the classes.

Client applications can remotely invoke other applications by sending globally (i.e., network-wide) recognized messages with parameters. Using the message names, as well as information about the classes of certain parameters and certain preference information, called context information, a reference to a specific method is selected from the data base. That method will perform the operation specified in the message. Other information in the data base is then used to locate and execute the actual code to implement the referenced method.

More particularly, in a data processing network containing a plurality of data processing platforms for executing applications, and a class data base having portions accessible to all of the data processing platforms, a process according to this invention of invoking an application to be executed on a server data processing platform from a client one of the applications executing on a client one of the data processing platforms comprises several steps, performed by the client data processing platform. The class data base includes a plurality of method entries each containing a reference to a corresponding mechanism for invoking a corresponding one of the applications to be run on a server one of the data processing platforms, and a plurality of class entries each containing information identifying a corresponding group of the method entries and identifying a different, uniquely identifiable class. Each of the classes is referenced by a different set of instances, instances being items that may be manipulated or accessed by the applications, and all the instances in each set having shared characteristics. The steps in the pprocess include receiving a message from the client application for the performance of a data processing operation involving a first instance; accessing the class data base, using the class referenced by the first instance and the received message, to select a first method entry and a corresponding mechanism for invoking a server one of the applications; and selecting, from among the data processing platforms, the server data processing platform to execute the server application. A data processing network according to this invention contains a plurality of data processing platforms for executing applications and comprises memory in the network containing a class data base having portions accessible to all of the data processing platforms, client ones of the data processing platforms having the capability for invoking a server application to be executed on the server data processing platform from a first client application executing on clients one of the data processing platforms, The class data base includesa plurality of method entries each containing a reference to a corresponding mechanism for invoking a corresponding one of the applications or procedures to be run on a server one of the data processing platforms, and a plurality of class entries each containing information identifying a corresponding group of the method entries and identifying a different, uniquely identifiable class, each of the classes identifying a different set of instances, instances being items that may be manipulated or accessed by the applications and all the instances in each set having shared characteristics. Client ones of the data processing platforms comprise means for receiving a message from the client application for the performance of a data processing operation involving a first instance; means, coupled to the receiving means, for accessing the class data base, using the class identifying the first instance and the received message, to select a first method entry and a corresponding mechanism for invoking a server one of the applications; and means, coupled to the accessing means, for selecting, from among the data processing platforms, the server data processing platform to execute the server application.

The accompanying drawings which are incorporated in and which constitute part of this specification, illustrate an implementation of the invention and, together with the description, explain the principles of the invention.

IV. BRIEF DESCRIPTION OF DRAWINGS

Figure 1 is a diagram of a network which can be used in a preferred implementation of the present invention.

Figure 2 is an illustration of the major components of an object-oriented model of this invention in relationship to an application.

Figure 3 is an illustration of the relationships between the components of the object-oriented

applications.

ACAS software components, 130, 230, and 330 implement the object-oriented approach of this invention. Preferably, ACAS software components 130, 230, and 330 consist of a number of software modules as described in greater detail below.

Operating systems 140, 240, and 340 are the standard operating systems which are tied to the corresponding CPUs 110, 210, and 310, respectively, of the platform 100, 200, and 300, respectively.

Memories 150, 250, and 350 serve several functions. One of the functions is, of course, to provide general storage for the associated platform. Another function is to store applications 120, 220, and 320, ACAS software components 130, 230, and 330, and operating systems 140, 240, and 340 prior to their execution by the respective CPU 110, 310, and 310.

In addition, portions of memories 150, 250, and 350 contain information for a network wide, or "global," data base which is shared and available directly or indirectly by all of the platforms 100, 200, and 300 in network 50. The global data base is described in greater detail below.

B. Elements of the Object-Oriented Architecture

(1) Definitions of the Elements

Object-oriented methods have been used in programming to separate the interface of data from actual implementation, but such methods have not been applied to heterogeneous networks. In the present invention, object-oriented techniques are used to separate the actual applications and their data from the implementation of operations on that data by other applications.

The object-oriented architecture of this invention preferably includes certain key elements. Figure 2 explains the relationship between certain of those elements and certain conventional features of applications. As shown in Figure 2, an application 260 can be described in two ways. First, that application has certain application definitions 265. For example, if the application 260 is a word processing program, then the application definitions could include definitions of what operations that word processing program can perform and what kind of data that word processing system can operate upon.

In addition, application 260 includes application data 268. Application data 268 is the specific data upon which application 260 operates.

In accordance with the present invention, the application data is not "handled" by the object-oriented architecture. Instead, the present invention is organized around characterizing the application definitions and the application data in terms of object types, as referred to in the remainder of this description as objects. Objects are not shown in Figure 2, but they pervade the elements that are shown.

In the discussion which follows, the term "object" will refer generally to several different kinds of elements, all of which have two characteristics in common. First, they refer to external capabilities, meaning that objects refer to or describe those portions of application definitions or application data which need to be communicated with other applications. Second, they are generic meaning that objects are intended to be available to all applications, and as such have a universally recognized and unique name for all applications that have interfaces to the objects. The present invention involves the handling of objects rather than the handling of specific data or applications.

As shown in Figure 2, two elements of the object-oriented architecture of this invention are developed from the application definitions 265. One is classes 270 and the other is methods 280. Classes are objects in the sense that the names of the classes and the features of the classes are both external and generic. Furthermore, classes can be used as means for describing not only applications, but also the data used by the applications.

In addition, one can derive certain types of operations from the application definitions 265 that are performed by that application, and these are specific examples of methods 280. Again, however, the specific methods 280 are not managed by the system, but rather can be organized into classes. The classes for those methods (called method objects) are generic and external, even through the specific commands or operations executed by the applications are not.

Instances 290, which are derived from the application data 268, are items that may be manipulated or accessed by an application. Again, the instances are not objects managed by this architecture. Instead, instances are organized into classes such that instances in the same classes share common characteristics. For example, a specific DECwrite application, which is a compound document editor, may be operating on a specific file called MYFILE. This is a specific file, and it is not handled by the ACAS system. Instead, MYFILE may belong to a class of compatible files, such as ASCII_FILE, which is generic and therefore a class object.

By the same token, a specific DECwrite application is not managed by the entire system. Instead, however, the specific DECwrite application may belong to a class called DECwrite which is generic and a class object.

As can be seen from Figure 2, applications can

that indicate that the methods, use a certain interaction type, and have a certain server start-up type.

Method object 410 is representative of the CUT function in EMACS applications. Associated with method 410 is a set of attributes 430 which includes those inherited from method object 400. Briefly, the PlatformType attribute indicates the platform on which the method object can be executed. The InteractionType attribute describes the actual type of method which will be executed within a particular method server. Examples of values for this attribute which are explained below, are: BUILT_IN, SCRIPT_SERVER and DYNAMIC_LOAD. The ServerStartupType attribute indicates an appropriate invocation mechanism to be used for the method server. Examples of values for this attribute, which are also explained below, are: SHELL, DYNAMIC_LOAD and NAMED_APPLICATION.

The set of attributes 430 specify that the associated methods operate on platforms which have an 80286 processor with the MS-DOS operating system, and have a BUILT_IN, interaction type, and a NAMED_APPLICATION server start-up type.

Similarly, method object 415, which is representative of the READ function in EMACS applications. Associated with method 415 is a set of attributes 435 which include those inherited from method object 400, but which also specify that the associated methods operate on VAX platforms running the VMS operating system, and have an interaction type of BUILT_IN, and a NAMED_APPLICATION server start-up type.

Method object 420 is a subclass of method object 400 representative of the READ function in EMACS applications. The attributes 440 for method class 420 have a platform type with a MIPS processor running the ULTRIX operating system with a BUILT_IN interaction type, and a NAMED_APPLICATION server start-up type.

Class 450, on the other hand, is a superclass of class 460 called FILES, and a class 465 called APPLICATIONS. Class 460 refers to data objects. As shown in Figure 4, class 460, which would have attributes (not shown), is a superclass of class 470. Class 470 is called ASCII_FILE. For example, class 470 could represent all the files within network 50 (see Figure 1) having the common characteristics of ASCII files. The common characteristics can be described in the attributes for class 470, which are not shown in Figure 4.

The class 470 would then be the class for several instances, but the instances are not shown in Figure 4 because they are not managed by the object-oriented architecture. What is shown in Figure 4 are the messages which the class 470 will support, and the only one shown for purposes of simplicity is the EDIT message 490.

A class supporting a message means that when the message is used as an interface into this object-oriented architecture, it can be used with the class that supports it, and therefore instances within that class. Thus, in the example shown in Figure 4, an EDIT message, can be sent to all instances in the ASCII_FILE class.

APPLICATIONS class 465 is also a superclass, and one of its subclasses, EDITOR class 475, is shown. EDITOR class 475 is a superclass to specific applications classes 480, 483 and 485, corresponding to WORD_A, WORD_B, or WORD_C. Each of the classes, such as WORD_C 485, represents a specific application, such as EMACS or TPU. Thus, each application is defined by one class. An application class may, however, refer to the implementation behavior of more than one application.

The application classes also support messages, which is shown by the message CUT 495 being supported by the application class 485. This reflects the fact that at the time of class definition, it was determined that any application represented by the class 485 would have to support a message CUT.

As mentioned briefly above, in the preferred implementation, applications are organized into a hierarchy of classes with a parent class, referred to as a superclass, and child classes referred to as subclasses. In Figure 4, class 465 is a superclass called EDITOR. All subclasses of this superclass would have at least the same set of particular unique characteristics or attributes of the superclass. In Figure 4, the subclasses of super class 475 EDITOR are WORD_A 480, WORD_B 483, and WORD_C 485. WORD_A might represent TPU applications, WORD_B 483 might represent all LSE applications, and WORD_C 485 might represent all EMACS applications. Each of these subclasses would have, in addition to the characteristics and attributes inherited from superclass 475, their own set of unique characteristics and attributes which differ in such a manner as to enable their separation as subclasses within the superclass 475 EDITOR.

In the preferred implementation of this invention, specific rules of inheritance allow for multiple inheritance among classes. This means that any subclass may have more than one superclass. Because this type of inheritance may create ambiguities at definition time, the superclasses are considered to be "ordered" at definition time to resolve potential inheritance conflicts. For instance, at the time of the definition of a subclass described below, if any conflicts arise due to the duplicate definition of a message or attribute in more than one of the listed superclasses, the message or

the method object ED_3_READ 550, with each of the executable codes being capable of performing the functions of the method object ED_3_READ 550 on a respective one of the platforms 100, 200, and 300. The system according to the preferred implementation includes a process which selects between the three executable codes.

Unlike the attributes 510 associated with the classes, the method attributes 560 of the class data base 500 associated with method objects 549 are used to locate and to execute an instance associated with a particular method object, such as method object 550, in the network. For purposes of simplicity, Figure 5 shows only one set of method attributes 561 in the class data base 500. The set 561 is associated with the method object 550 of the method objects 549 in the class data base 500. Although some of the method attributes in sets 560, can be arbitrarily specified by the users of the system and used by the system during execution, certain attributes are critical to the operation.

As shown in Figure 5, the method attributes in set 561 includes PlatformType = 80286.MS-DOS, InteractionType = BUILT_IN, and ServerStartup-Type = SHELL.

In the preferred implementation, two other method attributes are included in the method attribute set 561. One is an InvocationString attribute which defines an invocation string to be used in order to start the specified method server if it needs to be started. The value of this attribute must be a value appropriate for the particular platform specified in the first attribute. For example, if the value of the PlatformType attribute is MIPS.ULTRIX and the value of the ServerStartup-Type attribute is SHELL, then the value of this attribute should be an appropriate ULTRIX shell command.

D. Information Flow

Before discussing the details of the preferred implementation of this invention, the flow of information throughout the entire system will be explained with reference to Figure 6.

Figure 6 includes a diagram 600 showing different components of the network 50 shown in Figure 1 and the information flowing between those components. Applications 610 and 670 in Figure 6 each correspond to any one of the applications 120, 220, or 320, respectively, and the ACAS software components 620 and 660 each correspond to anyone of the ACAS software components 130, 230, or 330. The class data bases 640 and the context object data bases 630 are stored in one or more of the memories 150, 250, and 350.

As explained in greater detail below, an application 610, which will be referred to as a "client application," sends messages. The messages may include instance handles which are the mechanisms used to identify the client (or any other) application's instances. The messages are received by the ACAS software component 620 in the client platform.

ACAS software component 620 then uses the names of the messages and the classes of the instances referred to by the instance handles to find the method maps in class data bases 640. ACAS software component 620 may also use context information from context object data bases 630 to select a method identifier from the method map which identifier represents the method to be executed. The context information is also used to select a platform, called the "server platform," on which to execute the selected method. The context information will be described in detail below.

ACAS software component 620 sends the method identifier retrieved from the class data base 640 and the instance handles to an ACAS software component 660 in the server platform. Thereafter, the ACAS software component 660 takes the appropriate steps to execute the identified method using a "server application" 670 or informs the ACAS software component 620 that the server platform containing ACAS software component 660 cannot respond to the request. In this latter case, the ACAS software component 620 then reviews the context information to select another platform in the network as a server platform or else informs the client that the request has failed.

If the execution of the method identified in Figure 6 by the server application 670 generates a message to be returned to the client application 610, then that message along with additional information is passed from server application 670 to ACAS software component 660 in the server platform. ACAS software component 660 in the server platform then sends responses to ACAS software component 620 in the client platform, which relays those responses to the client application 610 in the client platform.

All these transactions will be described in greater detail below.

E. Memory Systems

(1) Global Class Data Base

A diagram of the entire memory system 700 is shown in Figure 7. Memory system 700 includes a global class data base 705 and local class data bases 710, 730 and 750. A network-wide memory 705 is also provided to make certain other information, described below, available to users of the network.

Global class data base 705 contains informa-

The other mechanism, instance naming, employs a standard for the naming of instances in the preferred implementation. The standard instance handle is a string represented by the following logical structure:

<class><storage_class><location><instance_reference_data>

The term "class" is the name of the associated ACAS class. The term "storage_class" is an alternative to the class name and is the name of the storage class. The term "location" is the logical location, such as the node, of the instance. The "location" is optional and will be used if a client desires a method to run at the same location as the instance is located. The term "instance_reference_data" is the application private portion of the instance handle.

Instance handles allow implementations to refer to instances abstractly, thereby avoiding the need to manage the instances themselves.

The instance handle preferably includes the class or storage class (if necessary), location of the instance, and the identifier for the instance. For example, in the message:

EDIT (INSTANCE_HANDLE)

EDIT represents the desired operation. The INSTANCE_HANDLE string could be ASCII_FILE/NODE_1/MYFILE.TXT. In this instance handle, ASCII_FILE represents the class, NODE_1 is the location of the instance, and MYFILE.TXT is the identifier of the instance. This message provides sufficient class and message information to find the proper method map. It will be apparent to those of ordinary skill in the art that other formats may be employed for the INSTANCE_HANDLE string to accomplish the same objectives as the preferred implementation does.

As explained above, all classes in a global class data base of the preferred implementation have unique names with the particular global class data base. The class name is generally assigned by the user who first defines the class.

(2) Local Class Data Bases

In addition to a global class data base, the preferred implementation also supports local class data bases for class and method definitions. The local class data bases function similar to the global class data base, except the contents of the local class data bases are not globally available. They need only be available for their local node. Thus, the local class data bases need not be distributed or replicated in other nodes.

Figure 7 shows a preferred implementation of the local class data bases 710, 730, and 750 in memories 150, 250 and 350, respectively. The local class data bases 710, 730 and 750 hold the class and method information created by the corresponding nodes which has not yet been added to the global class data base.

In the preferred embodiment, memories 150, 250 and 350 also contain node caches 720, 740 and 760, respectively, which hold method and class information loaded from global class data base 705. Caches are an optimization and are not strictly required.

The data base system used to implement the local class data base must provide name uniqueness within a single data base. Access control for the local class data base is only required at the data base level. The preferred implementation of a local class data base relies upon the underlying security mechanisms within the data base system to control access to the contents of the local class data base.

Use of the local class data base provides several advantages over use of the global class data base. For example, the local class data base provides the ability for applications on each node to continue to communicate with each other in an object-oriented manner even when the network is unavailable. In such a situation, applications on the node can continue to invoke other applications that are local to that node.

In addition, using a local class data base provides better performance for applications that reside in the same node as the local class data base because many invocations can be handled completely within the confines of a single platform. On platforms in which most applications will most likely use invocations that can be handled locally, use of the local class data base may eliminate or greatly reduce the need for network activity, such as accessing the global class data base, to accomplish an invocation.

The class data bases are preferably searched for class and method information by searching the local data bases before searching the global data base. The local data bases of each node are preferably searched in a predetermined order as explained below. As soon as the desired information is located, the search stops. Only if the desired information cannot be located in a local data base is the global data base searched. Thus the search order defines the "priority" of the class data bases.

Figure 8 shows one design of a portion of a local class data base 800. This design, however, is not critical to the invention. Preferably local class data base 800 contains a data base header 810 which is used to locate other organizational information in the local class data base 800 such as indices and allocation maps. Local class data base 800 also includes a block storage space 815 containing a number of blocks 820, 822, and 824

tributes. This facility also provides for the specification of inheritance among classes and, along with the LOADER/UNLOADER software component 1010 described above, can be used to modify existing definitions within the global class data base and the local class data base. In addition, the object definition facility preferably performs the necessary syntax checks of class definition input and method definition input used to create new class and method definitions within the global class data base.

A user of the object definition facility must specify certain information to create a class. This information preferably includes: a global class name and identifier; global names and identifiers (if any) of the superclasses of this class; messages supported by this class, along with their associated types of arguments (if any); method maps defined and the messages to which each map relates; and attributes defined for this class.

Each message is preferably specified by generating a structure including the name of the message, parameters supported by the message, and a corresponding method map. Each message structure is converted into two sets of values in the preferred implementation. One set of values includes the message name and the list of parameters supported by the message. The other set of values identifies a set of method objects that represent implementations of the message.

Method objects are defined within the network environment in the same manner as classes. The object definition facility of the preferred implementation, however, has special provisions for defining of method objects. The following information is specified when defining a method object; the global name and identifier of the method object; global names and identifiers of the superclasses of the method object; and metadata (i.e., descriptions of data) stored as the method attributes. The method definition also specifies the arguments and their types corresponding to the parameters in the message, and whether the method involves a parameter list. This parameter list represents the input required by the executable code (discussed below) capable of being invoked by the method.

(2) Method/Class Definition

In the preferred implementation, the loading of class and method definitions may either be done prior to run-time or dynamically during run-time. Classes and method objects may be accessible either locally on a node within the network (called "local definition") or globally from all platforms in the network (called global definition") Both local and global definition can be accomplished using the LOADER/UNLOADER software component 1010 or any other acceptable mechanism.

(3) Server Registration

The purpose of server registration is to find method servers which are available to service requests from messages. Method servers are the active (i.e., currently running) processes implementing the methods. A method server may involve execution of the code of a single application or of many portions of the code of one or more applications.

The registration of method servers is distinct from the definition of classes and method objects. Whereas the definition of classes and method objects is used to identify their presence in the system, the registration of method servers is used to track their status (i.e., availability). If a method server is not registered, it is not known to the system.

(4) Application Installation & Definition

Preferably, support mechanisms are provided for registering and installing applications in the network. The preferred implementation provides the ability to define applications and application fragments in the object-oriented model of classes, subclasses, messages and methods stored in a class data base. The definition of applications in this manner is critical to the operation of the interapplication communication performed by the preferred implementation of this invention. Specifically, the storage of classes, subclasses, messages and methods in a class data base permits an application, during run-time, to update the class data base and continue processing using the updated class data base without having to recompile and relink.

Applications are defined in the same manner as other classes. In fact, as explained above, an application is itself defined to be a particular kind of class.

Applications are installed on specific platforms in the manner required for the particular operating system on that platform. In the preferred implementation of this invention, application installation also requires some additional functions. For example, unless it has already been defined, an application must provide its own class definition which is defined as a subclass of the existing ACAS_APPLICATION.

Application installation may use class definitions already installed or may add new definitions. At application installation time, an installation procedure may compile and register the class definitions supported by the application into either a local class data base or the global class data base

invention is implemented using a client/server model in which a client generates requests and a server responds to requests. In the following discussion, the service or operation associated with a client application on a client platform is called the "client service," and the service or operation associated with a server application executing on a server platform is called a "server service." The client service and the server service of the preferred implementation rely upon a transport system which is capable of transmitting messages from the client platform to and from the server platform. In the preferred implementation, an RPC-like communications system is used as the transport system.

Each of the ACAS software components 130, 230, and 330 shown in Figure 1 preferably includes client service components and the server service components which represent the client and server services, respectively. This is shown, for example, in Figure 12 which is a diagram of two platforms 1200 and 1300 and a network bus 55. Platforms 1200 and 1300 can correspond to any of platforms 100, 200, or 300 in Figure 1.

Located in platforms 1200 and 1300 are memories 1250 and 1350, respectively, and CPUs 1210 and 1310, respectively. The elements in the platforms 1200 and 1300 function in the same manner as similar elements described above with reference to Figure 1. CPU 1210 executes a client application 1220 and CPU 1310 executes a server application 1320. CPUs 1210 and 1310 also execute OP SYS 1 1240 and OP SYS 2 1340, respectively, and ACAS software components 1230 and 1330, respectively.

ACAS software components 1230 and 1330 preferably include dispatcher software components 1232 and 1332, respectively, control server software components 1234 and 1334, respectively, invoker software components 1236 and 1336, respectively, and the auxiliary software components 1237 and 1337, respectively.

For the most part, invoker software components 1236 and 1336 represent the client service and dispatcher software components 1232 and 1332 represent the server service. The auxiliary software components 1237 and 1337 represent some other operations of the preferred implementation. Since platforms 1200 and 1300 in the network contain an invoker software component 1236 and 1336, respectively, a control server software component 1234 and 1334, respectively, and a dispatcher software component 1232 and 1332, respectively, either platform can act as a client or a server.

In the preferred implementation, the invoker software components 1236 and 1336 and the dispatcher software components 1232 and 1332 have the responsibility for interpreting class and method information in the class data bases, as well as context data in the context object data base, to determine the appropriate method to invoke, to determine how to invoke that method, and to dispatch the necessary commands to execute the code to implement the method. The invoker software components 1236 and 1336 and the dispatcher software components 1232 and 1332 also insulate client applications from the details of the method invocation and the transport level mechanisms.

The control server software components 1234 and 1334 have several functions. One function is to store information on currently running server applications on the platforms 1200 and 1300 in the network 50. The control server software components 1234 and 1334 also execute processes to start new applications that become method servers. Another function performed by control server software components 1234 and 1334 is method server registration. For example, the control server software component 1334 stores information regarding the method server, identified by the server application 1320, currently running on the server platform 1300. The control server software component 1334 also communicates with the server registration facility in network-wide memory 704 (Figure 7) to store status information regarding the server application 1320.

The auxiliary software components 1237 and 1337 represent operations of the ACAS software components 1230 and 1330 such as class and method object definition and registration, method executable registration (described below) in a method executable catalog of each platform, and functions of the LOADER/UNLOADER software component 1010 (Figure 10).

For purpose of the following discussion, the platform 1200 is referred to as the client platform and the platform 1300 is referred to as the server platform. In this example, the client application 1220 communicates with the server application 1320 in the server platform 1300 in an objected-oriented fashion. It is also possible in accordance with the present invention and in the preferred implementation for a client application on one platform to communicate with a server application on the same platform.

When the client application 1220 communicates with the server application 1320, the dispatcher software component 1232 and control server software component 1234 of the client platform 1200 is not involved, and are therefore shaded in Figure 12. Likewise, invoker software component 1336 of the server platform 1300 is shaded because it is not active.

Figure 13 is a flow diagram 1360 outlining the major functions performed in an invocation of a

method server, such as the server application 1320, that has made itself known to the network 50 as being already started.

If there is a running method server, the invoker software component also queries server platform tables of the context object data base 630, to determine the location of a remote platform in the network 50 (Figure. 1) which the user of the client application 1220 would prefer to execute the method of invocation request processed by the invoker software component 1236. If however, the server application 1320 is not available, the control server software component 1334 notifies the invoker software component 1236 that the server application is not available on the selected remote platform. The invoker software component 1236 processing outlined above begins again with querying the server platform table of the context object data bases 630 and server registration facility 1420 to select another platform in the network 50 upon which to execute the identified method.

Next, the invoker software component 1236 transmits a query to the control server software component 1334 of the preferred server platform which causes control server software component 1334 to query a control server registry 1425 to determine whether the desired method server on the preferred server platform is available to process the method identified in the processed method invocation request. Availability of a method server is determined in the preferred implementation by examining in the control server registry 1425 to find out whether the method server is currently able to process methods invoked by client applications.

If the control server software component 1334 indicates to the invoker software component 1236 that the method server, in the form of server application 1320, is available, the invoker software component 1236 transmits the processed method invocation request to the dispatcher software component 1332 of the server platform. The invoker software component 1236 can also transmit information from the context object data base 630, which can then be used by the desired method server.

The dispatcher software component 1332 then begins to process the desired method. This process, referred to as the "dispatching process," generally involves dispatching the method identifier to begin the execution of the method by the server application 1320.

If, however, the server registration facility 1420 does not indicate that any copies of server application 1320 and currently running on a platform in the network, then the invoker software component 1236 may transmit a request to the control server software component 1334, using the information retrieved from the context object data bases 630 and the class data bases 640, to start the server application 1340. After the server application 1320 is started, the control server software component 1334 notifies the server registration facility 1420 to update the network-wide memory 704 (Figure 7) to indicate that the server application 1320 is running. Control server software component also updates the control server registry 1405 to indicate that the server application 1320 is available. The invoker software component 1236 then transmits the processed method invocation request to the dispatcher software component 1332 to execute the method corresponding to the method identifier of the processed method invocation request.

After the server application 1320 has completed its processing, it returns any output information requested by the processed method invocation request to the dispatcher software component 1332. The dispatcher software component 1332 then returns a response, as describe above, to the invoker software component 1236 along with any output information mapped into the output arguments of the processed method invocation request received by the dispatcher software component 1332.

(2) Invoker Operation

The portion of the process of method invocation performed by the invoker software component 1236 can now be described in greater detail. Preferably, that portion consists of several distinct phases including determining the proper method to be invoked (method resolution), server connection or start-up, and transport level communications to enable the dispatching of an identifier to the proper method to be executed by the method server or other executable code.

Figures 15A - 15D and 16 contain flow diagrams of procedures performed or called by the invoker software component 1236 of Figures 12 and 14. The main control procedure 1550 in Figures 15A - 15D represents the steps 1370, 1375, and 1380 (Figure 13) performed by invoker software component 1236.

As with procedure 1360, prior to entering the main control procedure 1550, the client application 1220 (Figures 12 and 14) is running normally without a method invocation request, and the ACAS software component 1230 is in a "wait" state. When the client application 1220 generates a method invocation request using the InvokeMethod procedure call, the main control procedure 1550 begins (step 1552 in Figure 15A) with the invoker software component 1236 receiving the method invocation request (step 1555).

The invoker software component 1236 first vali-

1565 in Figure 15B), for example, the value of the InteractionType method attribute is "BUILT_IN," then a check is made for an activation error (step 1566). If there was one, an error message is generated (step 1576) and control is returned to client application 1220 (step 1599 in Figure 15D).

If there was no error, a success flag is generated (step 1567), and the resolved method is executed by code already resident in the client application 1220 (step 1569).

If the method attributes do not indicate that the method is already linked to the client application 1220 (step 1565 in Figure 15B), invoker software component 1236 asks whether the method attributes indicate that the method is dynamically loadable (step 1570). Dynamically-loadable methods represent method executables which may be merged with executable code of client applications at run-time. Those skilled in the art will recognize that a dynamically-loadable method might be a method executable identified by a subprocedure or function of a client application. Preferably the test for a dynamically-loadable method server is accomplished by determining whether the value of the InteractionType method attribute is "DYNAMIC_LOAD." If so, then the invoker software component 1236 attempts to load the executable code identified by the resolved method into the client application 1220 (step 1572).

If an error occurred during the loading of the executable code (step 1574), then the invoker software component 1236 generates a message indicating that a load error occurred (step 1576) and returns the load error message to the client application 1220 (step 1599 in Figure 15D).

Otherwise, if there was no load error (step 1574), then the invoker software component 1236 then generates a flag indicating the successful completion of the method invocation (step 1567). Next, the dynamically loaded executable code corresponding to the resolved method is executed (step 1569), and control returns to the client application 1220 along with any output arguments (step 1599 in Figure 15D). Any errors in executing linked or dynamically-loadable method servers are preferably returned as parameter values.

If the method attributes do not indicate a previously-linked or dynamically-loadable method (steps 1565 and 1570 in Figure 15B), then the invoker software component 1236 must locate a running method server on a platform in the network that can handle the resolved method as described above with regard to Figure 14.

If the information retrieved from the server registration facility 1420 (step 1578) indicates that there is at least one running method server capable of performing the method identified by the resolved method (step 1579 in Figure 15C), then the invoker software component 1236 compares the information retrieve from the server registration facility 1420 with the entries on the server node table retrieved from the context object data bases 630 during the method resolution procedure 1600 to select a server platform in the network (step 1580).

Having selected a server platform, the invoker software component 1236 then transmits a QueryServer call to the control server software component 1334 of the selected server platform (step 1581). The functioning of the control server software component 1334 is described in detail below in connection with Figures 17A and 17B. Briefly, control server software component 1334 determines whether the desired method server is available or not.

The main control procedure 1550 of the invoker software component 1236 then continues in step 1582 (Figure 15C) by receiving a message generated by the control server software component 1334 about the desired method server's availability and translating the message into a format recognizable by the client platform. The invoker software component 1236 determines from the control server software component 1334 whether the method server corresponding to the resolved method is available to process the method identified by the resolved method (step 1583). If the corresponding method server is available, then processing of the invoker software component continues on Figure 15D by asking whether the method server is an asynchronous method server (step 1593) in Figure 15D. Asynchronous method servers are known in the art.

If the method server is asynchronous (step 1593), then the control server software component 1334 is called using the SignalServer call to signal the method server (step 1594). If the method server is not asynchronous (step 1593), or after an asynchronous method server is signaled (step 1594), the processed method invocation request, including the identifier for the method and information retrieved from the context object data bases during method resolution, is packed into a data structure used for communication in the network (step 1595) and the invoker software component 1236 then transmits the packed and processed method invocation request to the dispatcher software component 1332. The processes of the dispatcher software component 1332 will be described below with reference to Figures 18A and 18B.

After the dispatcher software component 1332 completes its processing and transmits a packed response, the invoker software component 1236 receives the packed response (step 1597), unpacks the response (step 1598), and returns the response to the client application 1220 to complete its processing (step 1599).

tion of this invention can operate both with applications written to take advantage of the features of this invention, or previously-written applications that have been modified for us with the preferred implementation. In so writing or modifying asynchronous applications to operate with the preferred implementation, a user includes program code that, in part, recognizes these asynchronous signals and, as described below, registers these signals and the following processed method invocation requests in queue. These operations are described below with reference to the processes performed by the dispatcher software component 1332.

If no other function has been requested, the control server software component 1334 determines whether the control server message indicates that the invoker software component 1236 is requesting that a new application, which resides on the same platform as the control server software component 1334, should be started to become a method server to process a method (step 1740 in Figure 17B). If so, then the control server software component 1334 checks the control server registry 1425 (step 1745) to determine whether the method executable of the new application, corresponding to the resolved method, resides on the selected platform (step 1750).

Control server registry 1425 has a local scope so that only the server platform 1300 is aware of resident method executables. The registration of method executables in registry 1425 involves registration of the actual executable code in executable files, for example shell scripts, that implement a method, and the status of those method executables. These items preferably have only a local registration scope because it is not necessary to manage the executable code globally.

If the corresponding method executable is identified in the control server registry 1425, then the selected platform can be a server platform. The control server software component 1334 executes a process to start the corresponding method executable and registers the resulting method server with the server registration facility 1420 and with the control server registry 1425 to indicate that the newly started method server is both running and available (step 1752). During this starting process, the control server software component 1334 also creates a context object data base capable of being used by the started method server. Next the control server software component 1334 then generates a message indicating that the application corresponding to the resolved method has been started and is now a method server (step 1754). This message is then transmitted to the invoker software component 1236 that requested that the method server be started (step 1790), and the control server software component 1334 has completed its processing (step 1799).

If the method executable corresponding to the resolved method is not identified in the control server registry 1425, then the control server software component 1334 generates an appropriate message indicating that the method executable was not started (step 1756). This message is then transmitted to the invoker software component 1236 that requested that the method server be started (step 1790), and the control server software component 1334 has completed its processing (step 1799).

If no other function has been requested, the control server software component 1334 determines whether the control server message is a request from the invoker software component 1236 for information concerning the availability of a running method server to execute a method identified by the resolved method (step 1760). If not, the control server software component 1334 generates an error message (step 1780), transmits that message to the invoker software component 1236 (step 1790), and completes its processing (step 1799).

Otherwise the control server software component 1334 retries the requested information on the running method server from the control server registry 1425 (step 1765). If the information from the control server registry 1425 indicates that the method server identified by the resolved method is available (step 1770), then the control server software component 1334 generates a message indicating the method server's availability (step 1775). This message is then transmitted to the invoker software component 1236 (step 1790), and the processing of the control server software component is complete (step 1799).

If, however, the control server registry 1425 indicates that the method server is not available (step 1770), then the control server software component 1334 generates a message indicating the unavailability of the method server (step 1777). The control server software component 1334 then transmits the generated message to the invoker software component (step 1790), and the processing of the control server software component 1334 is complete (step 1799).

(4) Dispatcher Operation

The process of dispatching method servers consists of dispatching methods to be processed by method servers and transport level communications. The dispatcher software component 1332 also handles different types of method invocations.

Asynchronous method invocations do not require that the client application wait for the identified method server to complete processing. For example, the invocation request can be placed on

quests on the queue to be processed by the asynchronous method server (step 1870 in Figure 18B). If there are no method invocation requests on the queue (step 1870), then the dispatcher processing is complete (step 1899).

If there were asynchronous method invocation requests on the queue (step 1870), the dispatcher software component 1332 takes the next asynchronous method invocation request off of the queue (step 1875). If the request taken off of the queue is invalid (step 1880), such as a request that cannot be processed by the method server, then processing returns to find out whether there are other queued method invocation requests (step 1870).

If the request taken of the queue is valid (step 1880), then the dispatcher software component 1332 dispatches the asynchronous method invocation request taken off the queue to be processed by the asynchronous method server (step 1885).

The question is then asked whether an error occurred in the processing of the method server (step 1887). The error, if any, is recorded (step 1888) then, or if an error did not occur, the dispatcher software component 11332 checks the queue (step 1870). In this manner all asynchronous invocation requests on the queue are processed, in turn, without blocking the client application that originated the method invocation request.

I. Summary

The present invention thus provides an efficient and simple manner for an application on one platform to invoke an application on the same or a different platform without needing to know details about the other platform, or even about the other application. Such invocation can even take place between unlike platforms in a heterogeneous network.

Because, in accordance with the object-oriented techniques of this invention, the data (or instances) and applications are not managed, those data and applications can be managed in the manner chosen by the application developers. By managing only objects and references to applications instead, the requirements on system resources are reduced, and the flexibility of the system is increased.

Another advantage afforded by this invention is the ability to invoke applications which are not currently running automatically. This further enhances the power and flexibility of systems implemented in accordance with this invention.

Persons of ordinary skill will recognize that modifications and variations may be made to this invention without departing from the spirit and scope of the general inventive concept. This invention in its broader aspects is therefore not limited to the specific details or representative methods shown and described.

Claims

1. In a data processing network containing
   a plurality of data processing platforms for executing applications, and
   a class data base having portions accessible to all of the data processing platforms, the class data base including
   a plurality of method entries each containing a reference to a mechanism for invoking a corresponding one of the applications to be run on a server one of the data processing platforms, and
   a plurality of class entries each containing information identifying a corresponding group of the method entries and identifying a different, uniquely identifiable class, each of the classes being referenced by a different set of instances, instances being items that may be manipulated or accessed by the applications, and all the instances in each set having shared characteristics,
   a process of invoking an application to be executed on the server data processing platform from a client one of the applications executing on a client one of the data processing platforms comprising the steps, performed by the client data processing platform, of:
   receiving a message from the client application for the performance of a data processing operation involving a first instance;
   accessing the class data base using the class referenced by the first instance and the received message to select a first method entry and a corresponding mechanism for invoking a server one of the applications; and
   selecting, from among the data processing platforms, the server data processing platform to execute the server application.

2. The process of claim 1 wherein the step of accessing the class data base includes the substep of:
   determining, from the class data base, the class identifying the first instance.

3. The process of claim 1 wherein the process further comprises the step of:
   receiving, after the execution of the server application by the server data processing platform, a response from the server application.

4. The process of claim 3 wherein the step of receiving the response includes the substep of:

able that meets the requirements of the method definition, includes the substep of

communicating with the method server registration facility to determine whether selected server application is loaded.

15. The process of claim 14 wherein the selected server processing platform includes a control server which maintains information corresponding to the server applications currently being executed on the selected server platform, and

wherein the substep of communicating with the method server registration facility includes the substep of

communicating with the control server of the selected server data processing platforms to determine whether the server application can be invoked.

16. The process of claim 15 wherein the control server starts the server application, and

wherein the substep of communicating with the control server further comprises the substep of

starting the server application.

17. The process of claim 14 wherein the data processing network further comprises

a transport service for performing network communication functions for communication between the data processing platforms, and

wherein the substep of communicating with the control server includes the substep of

querying the control server using the transport service to determine the availability of selected server data processing platforms

18. The process of claim 15 wherein the data processing network further comprises

a transport service for performing network communication functions for communication between the data processing platforms, and

wherein the substep of starting the server application includes the substep of

starting the server application using the transport service.

19. A data processing network containing a plurality of data processing platforms for executing applications comprising:

memory in the network containing a class data base having portions accessible to all of the data processing platforms, the class data base including

a plurality of method entries each containing a reference to a corresponding mechanism for invoking a corresponding one of the applications or procedures to be run on a server one of the data processing platforms, and

a plurality of class entries each containing information identifying a corresponding group of the method entries and identifying a different, uniquely identifiable class, each of the classes identifying a different set of instances, instances being items that may be manipulated or accessed by the applications and all the instances in each set having shared characteristics,

client ones of the data processing platforms having the capability for invoking a server application to be executed on the server data processing platform from a first client application executing on a client one of the data processing platforms, the client one of the data processing platforms comprising

means for receiving a message from the client application for the performance of a data processing operation involving a first instance;

means, coupled to the receiving means, for accessing the class data base, using the class identifying the first instance and the received message, to select a first method entry and a corresponding mechanism for invoking a server one of the applications; and

means, coupled to the accessing means, for selecting, from among the data processing platforms, the server data processing platform to execute the server application.

20. The data processing network of claim 19 wherein the means for accessing the class data base of the client data processing platforms includes

means for determining, from the class data base, the class identifying the first instance.

21. The data processing network of claim 19 wherein the client data processing platforms further comprise

means, coupled to the selecting means, for receiving a response from the server application.

22. The data processing network of claim 21 wherein the means for receiving the response from the server application of the client data processing platforms includes

means for receiving a parameter from the server application.

23. The data processing network of claim 19 further including

means for determining whether the message has any errors.

24. The data processing of claim 19 wherein the

a method server registration facility to track which of the server applications have been invoked and are executing, and

wherein the means for determining whether a server data processing platform is available that meets the requirements of the method definition includes

means for communicating with the method server registration facility to determine the availability of selected server data processing platforms.

36. The data processing network of claim of 35 wherein the selected server processing platforms each include

a control server which maintains information corresponding to the server applications currently being executed and starts the selected server application, and

wherein the communicating means includes

means for querying the control server of the selected server data processing platforms to determine the availability of selected server data processing platforms.

37. The data processing network of claim 36 wherein the means for querying the control server further comprises:

means for starting the server application using the reference to a corresponding mechanism.

38. The data processing network of claim 34 wherein the data processing network further comprises

a transport service for performing network communication functions for communication between the data processing platforms, and

wherein the means for querying with the control server includes

means, coupled to the transport service, for examining the control server of the selected server data processing platforms to determine the availability of selected server data processing platforms.

39. In a client platform connected to a data processing network comprising

a plurality of data processing platforms which permit remote invocation of server applications located on server ones of the data processing platforms by client applications located on client ones of the data processing platforms, and

a class data base containing

a plurality of method entries containing references to invocation mechanisms to invoke the server applications, and

a plurality of class entries each containing information for a different, uniquely identifiable class, the classes identifying types of the instances, which are items that may be manipulated or accessed by the applications according to shared characteristics, each of the class entries corresponding to groups of the method entries,

a process of remotely invoking a server application comprising the steps, performed by a client one of the data processing platforms, of:

receiving a message from the client application for the performance of a first method involving a first instance;

determining the class identifying the first instance;

accessing the class data base using the determined class and the received message to select the first method entry and a reference to a corresponding mechanism to find a server one of the data processing platforms capable of executing the server application;

invoking, using the transmission of the reference, the execution of the server application by the first server data processing platforms to execute the first method on the first instance; and

receiving a response from the server data processing platform.

40. The process of claim 39 wherein the first method entry includes

method definitions containing metadata characterizing the corresponding method entry, and

wherein the step of accessing the class data base includes the substep of

analyzing the metadata in the first method entry.

41. The process of claim 40 wherein the data processing network includes a transport service to provide communication between the data processing platforms,

wherein the metadata in selected ones of the method entries indicates whether the server application requires use of the transport service to be invoked, and

wherein the step of accessing the class data base further includes the substep of:

determining whether the method definition for the first method indicates that the first method is to be performed using the transport service to communicate with the server data processing platform.

42. The process of claim 41 further comprising the

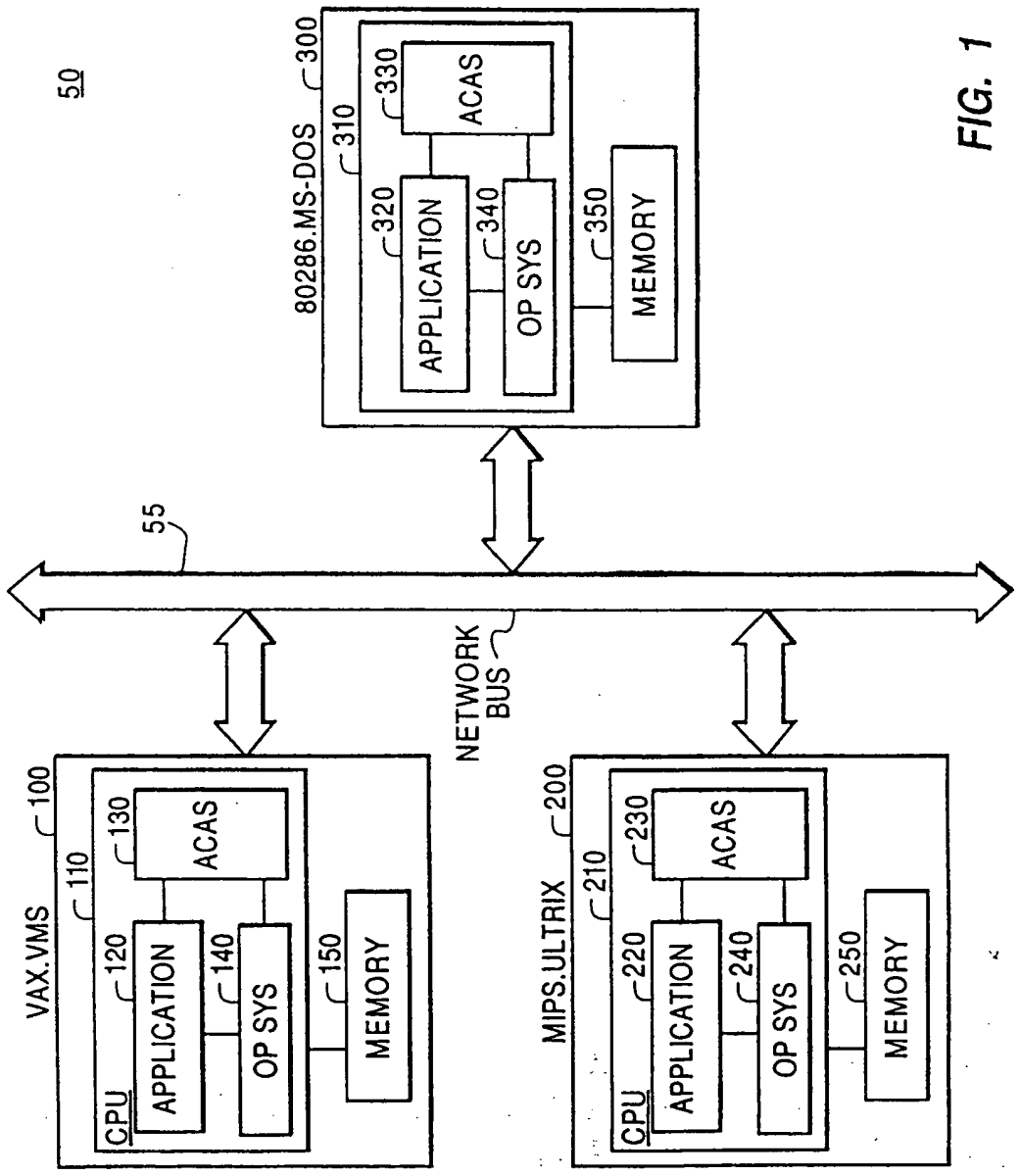
50
55
NETWORK BUS
80286.MS-DOS
300
310
320
APPLICATION
330
ACAS
340
OP SYS
350
MEMORY
VAX.VMS
100
110
120
APPLICATION
130
ACAS
140
OP SYS
150
MEMORY
CPU
MIPS.ULTRIX
200
210
220
APPLICATION
230
ACAS
240
OP SYS
250
MEMORY
CPU

FIG. 1

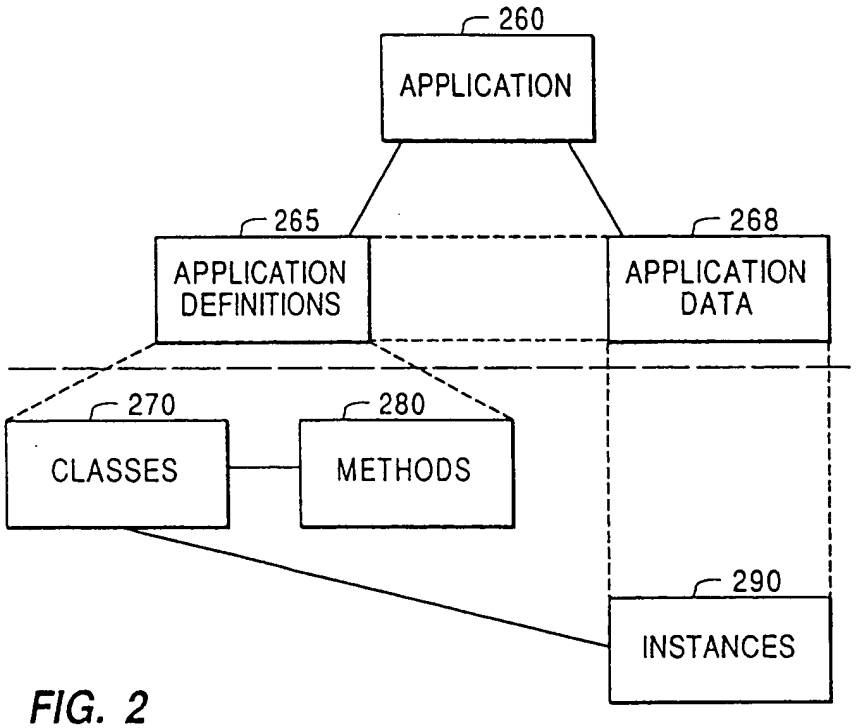
260
APPLICATION
265
APPLICATION DEFINITIONS
268
APPLICATION DATA
270
CLASSES
280
METHODS
290
INSTANCES

FIG. 2

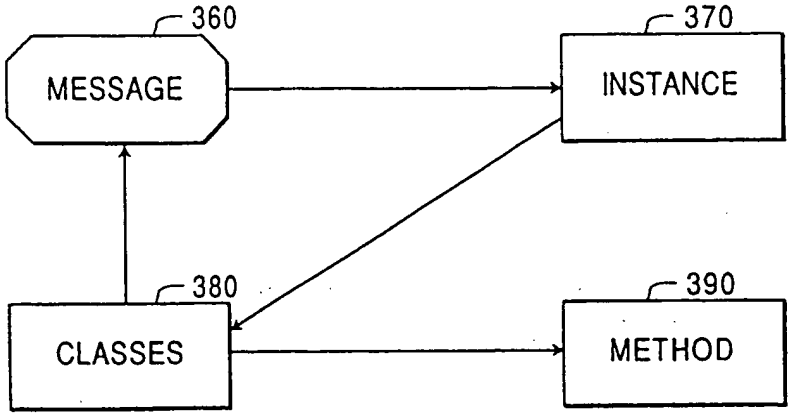
360
MESSAGE
370
INSTANCE
380
CLASSES
390
METHOD

FIG. 3

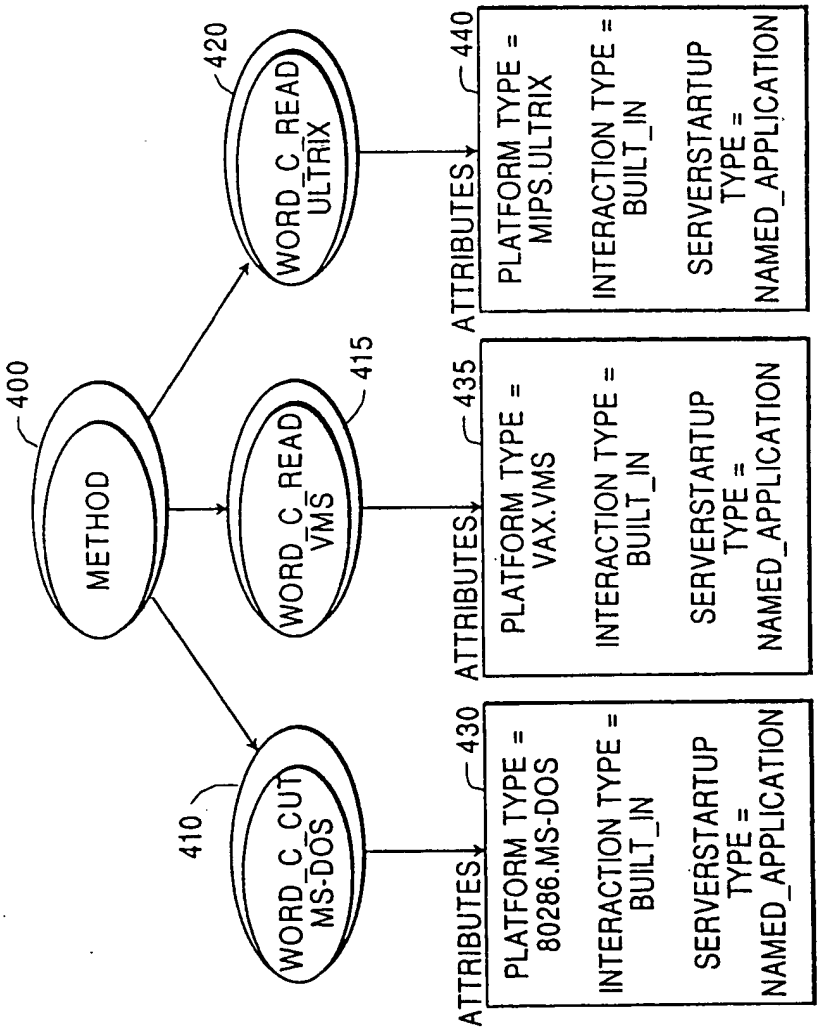
400
METHOD
410
WORD_C_CUT
MS-DOS
415
WORD_C_READ
VMS
420
WORD_C_READ
ULTRIX
ATTRIBUTES
430
PLATFORM TYPE = 80286.MS-DOS
INTERACTION TYPE = BUILT_IN
SERVERSTARTUP TYPE = NAMED_APPLICATION
ATTRIBUTES
435
PLATFORM TYPE = VAX.VMS
INTERACTION TYPE = BUILT_IN
SERVERSTARTUP TYPE = NAMED_APPLICATION
ATTRIBUTES
440
PLATFORM TYPE = MIPS.ULTRIX
INTERACTION TYPE = BUILT_IN
SERVERSTARTUP TYPE = NAMED_APPLICATION

FIG. 4A

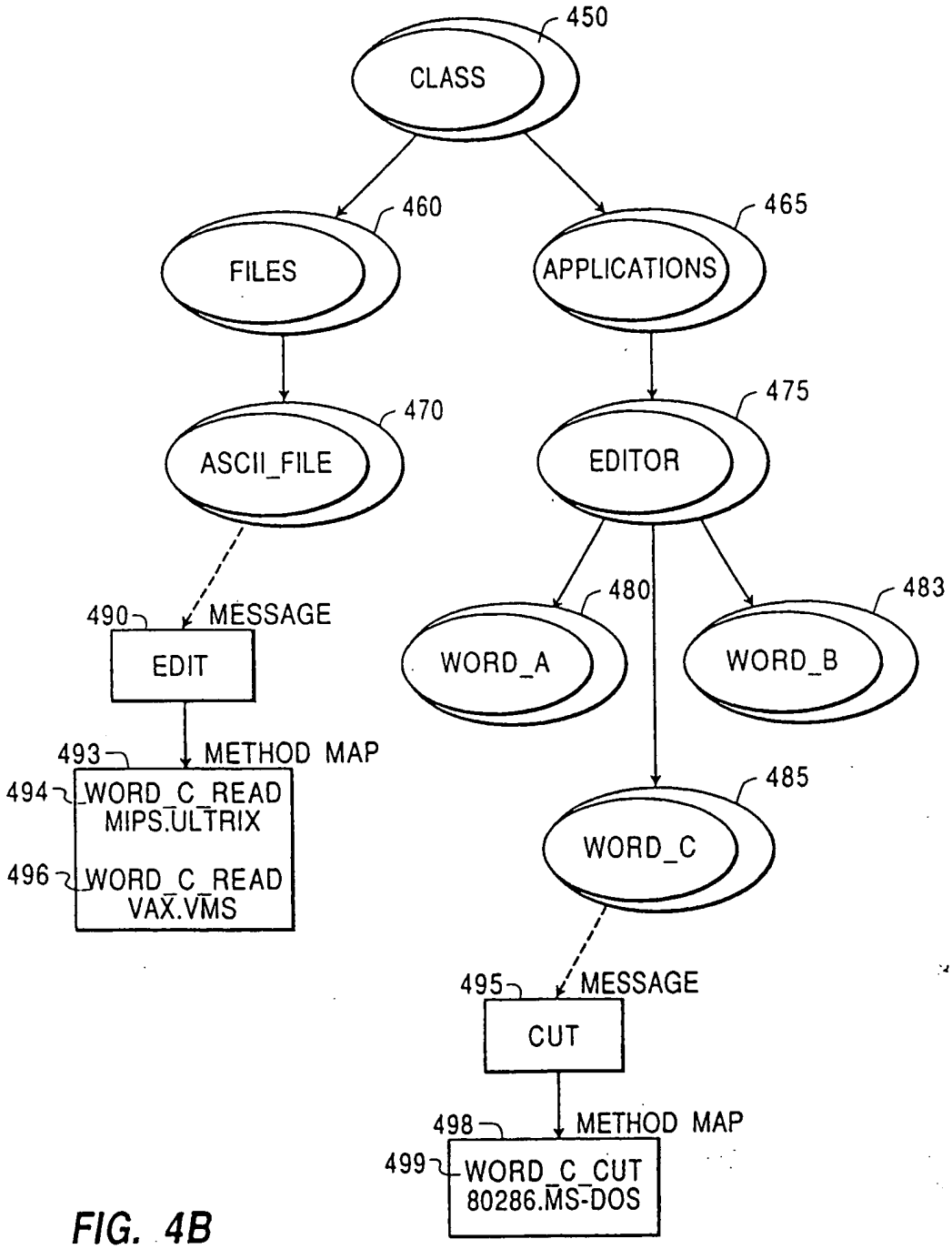
450
CLASS
460
FILES
465
APPLICATIONS
470
ASCII_FILE
475
EDITOR
490
MESSAGE
EDIT
480
WORD_A
483
WORD_B
493
METHOD MAP
494
WORD_C_READ
MIPS.ULTRIX
496
WORD_C_READ
VAX.VMS
485
WORD_C
495
MESSAGE
CUT
498
METHOD MAP
499
WORD_C_CUT
80286.MS-DOS

FIG. 4B

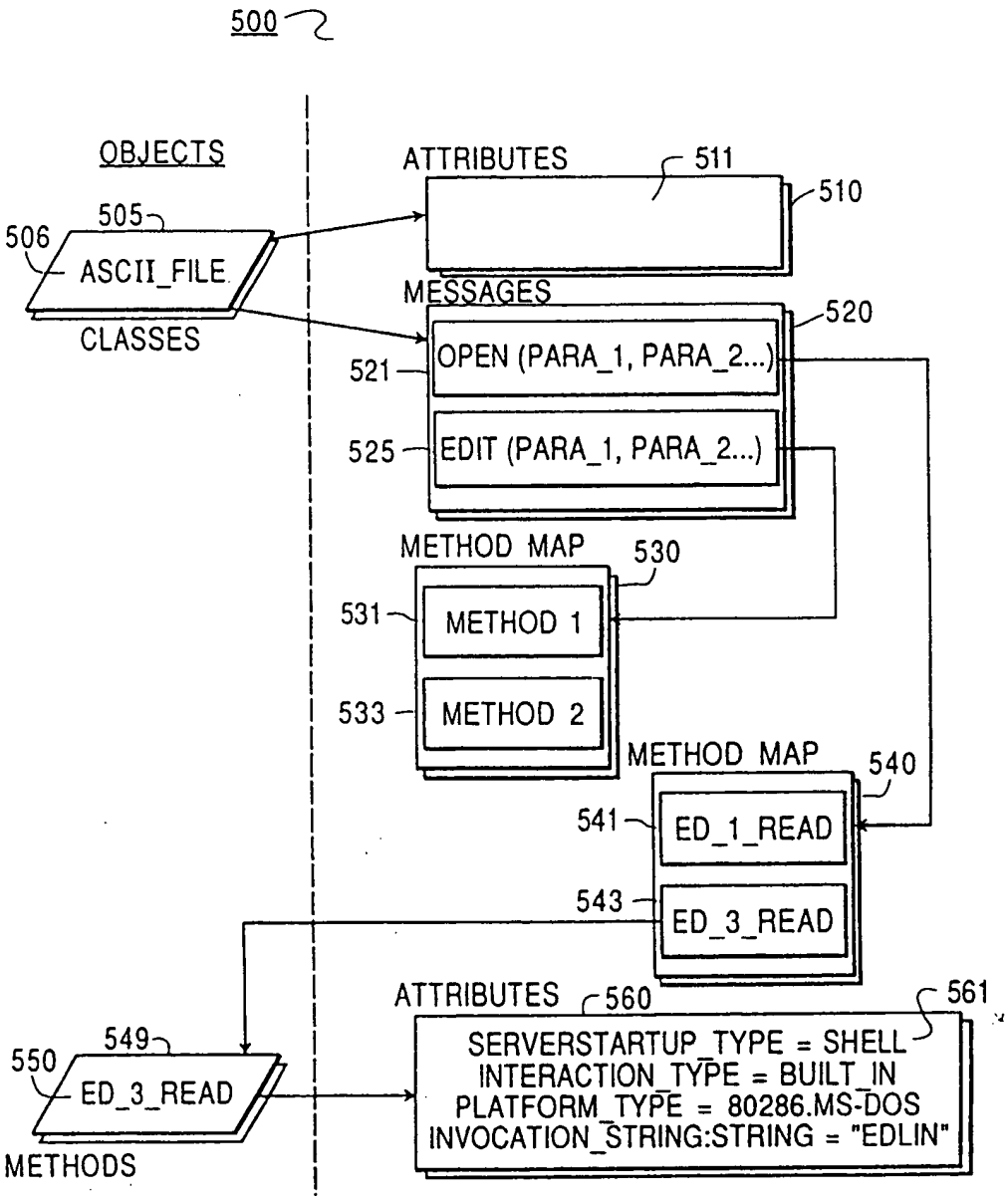
500
OBJECTS
ATTRIBUTES
511
510
505
506
ASCII_FILE
CLASSES
MESSAGES
520
OPEN (PARA_1, PARA_2...)
521
EDIT (PARA_1, PARA_2...)
525
METHOD MAP
530
531
METHOD 1
533
METHOD 2
METHOD MAP
540
541
ED_1_READ
543
ED_3_READ
ATTRIBUTES
560
561
SERVERSTARTUP_TYPE = SHELL
INTERACTION_TYPE = BUILT_IN
PLATFORM_TYPE = 80286.MS-DOS
INVOCATION_STRING:STRING = "EDLIN"
549
550
ED_3_READ
METHODS

FIG. 5

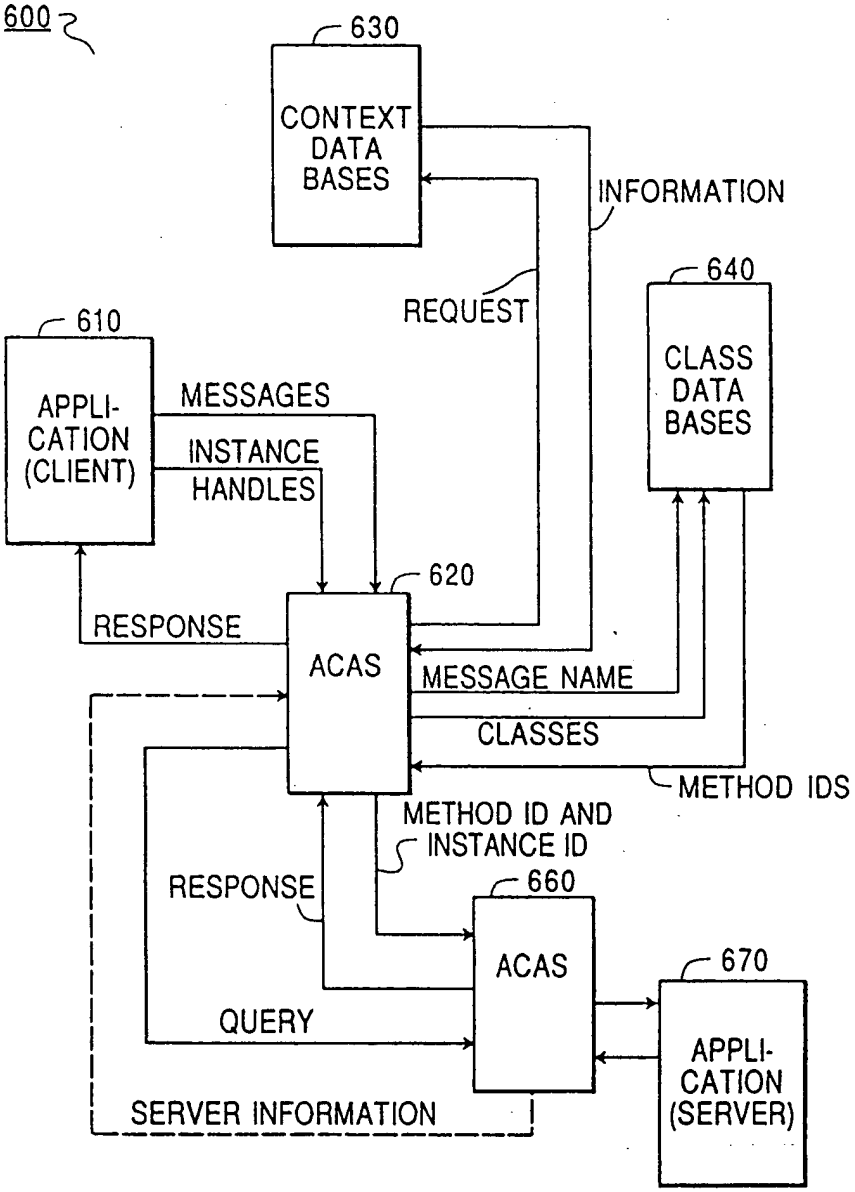
600
630
CONTEXT DATA BASES
INFORMATION
REQUEST
640
CLASS DATA BASES
610
APPLI-CATION (CLIENT)
MESSAGES
INSTANCE HANDLES
620
RESPONSE
ACAS
MESSAGE NAME
CLASSES
METHOD IDS
METHOD ID AND INSTANCE ID
RESPONSE
660
ACAS
670
APPLI-CATION (SERVER)
QUERY
SERVER INFORMATION

FIG. 6

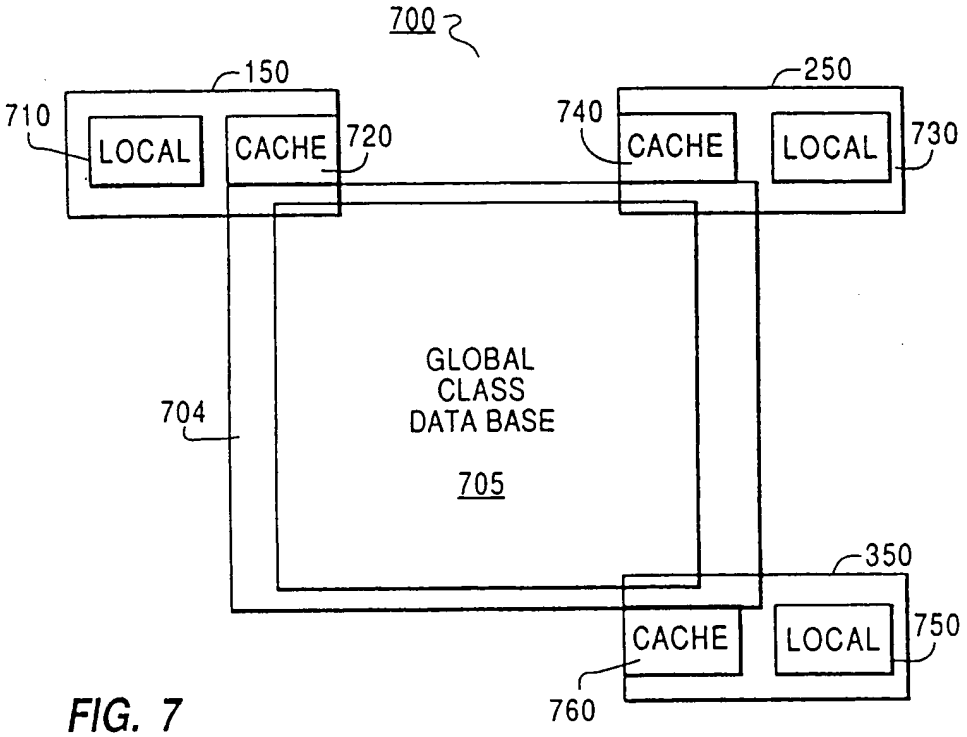
700
150
250
710
LOCAL
CACHE
720
740
CACHE
LOCAL
730
GLOBAL
CLASS
DATA BASE
705
704
350
CACHE
LOCAL
750
760

FIG. 7

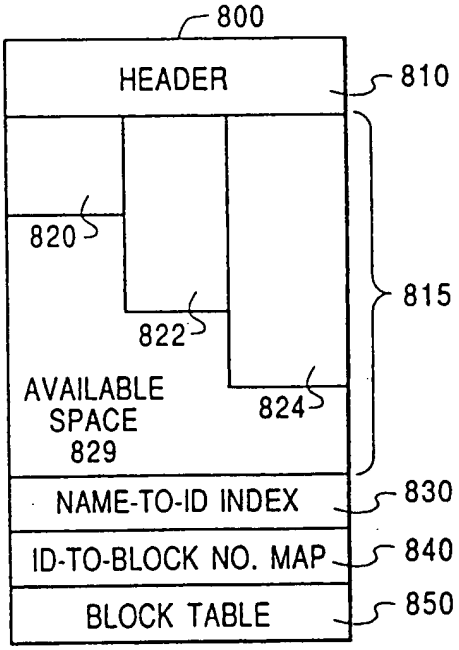
800
HEADER
810
820
822
824
815
AVAILABLE
SPACE
829
NAME-TO-ID INDEX
830
ID-TO-BLOCK NO. MAP
840
BLOCK TABLE
850

FIG. 8

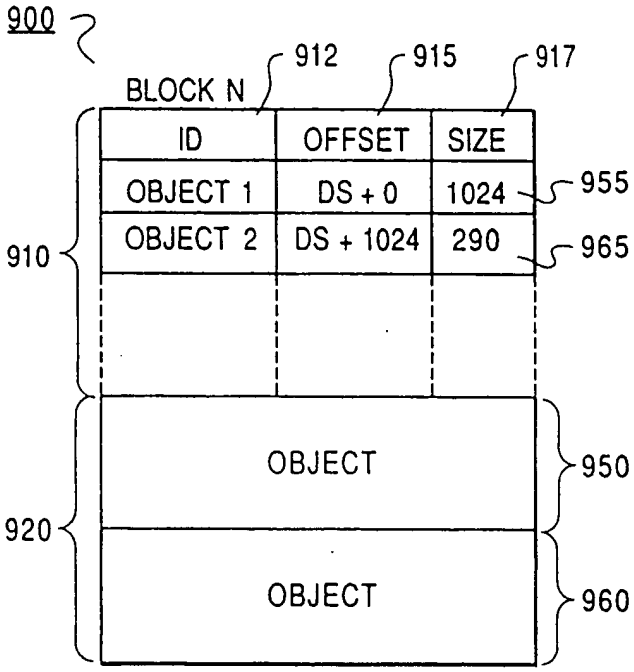
900
912
915
917
BLOCK N
ID
OFFSET
SIZE
OBJECT 1
DS + 0
1024
955
OBJECT 2
DS + 1024
290
965
910
OBJECT
950
920
OBJECT
960

FIG. 9

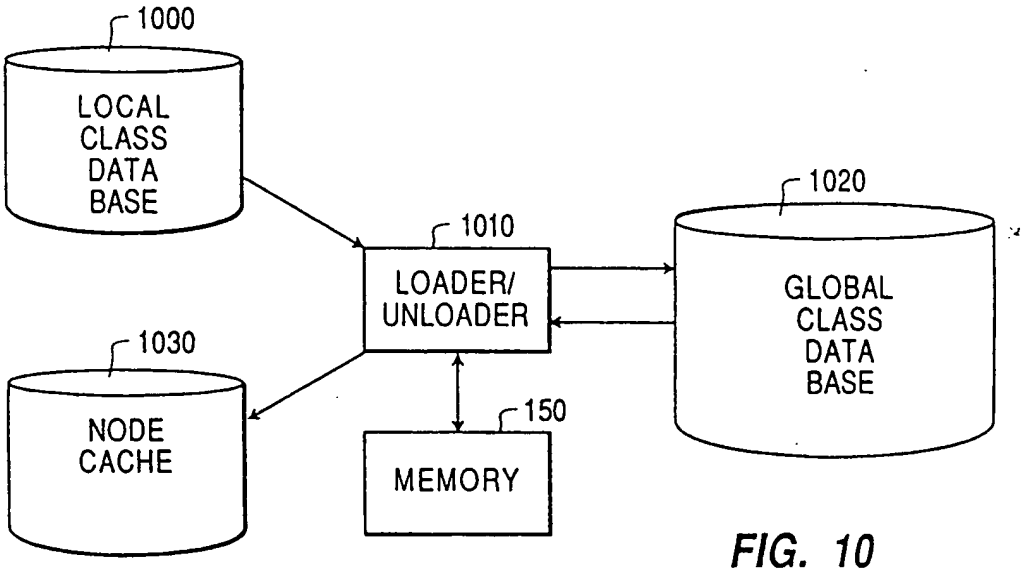
1000
LOCAL CLASS DATA BASE
1010
LOADER/ UNLOADER
1020
GLOBAL CLASS DATA BASE
1030
NODE CACHE
150
MEMORY

FIG. 10

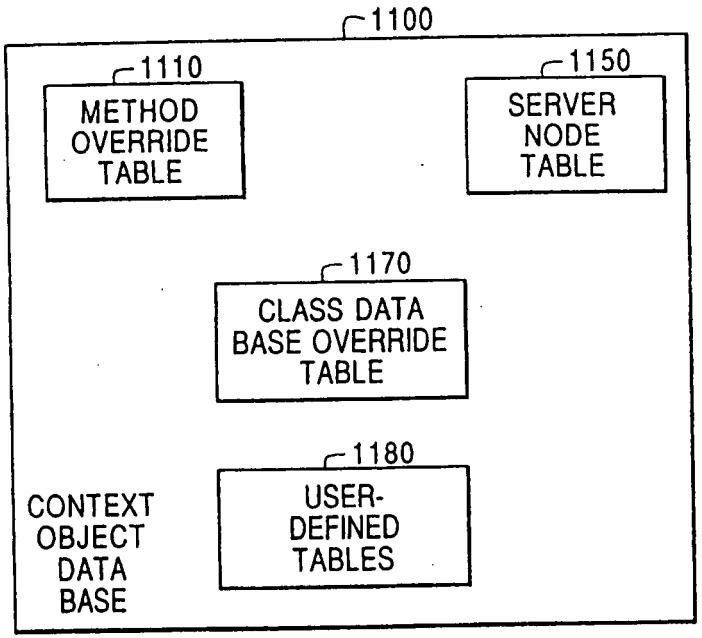

FIG. 11A

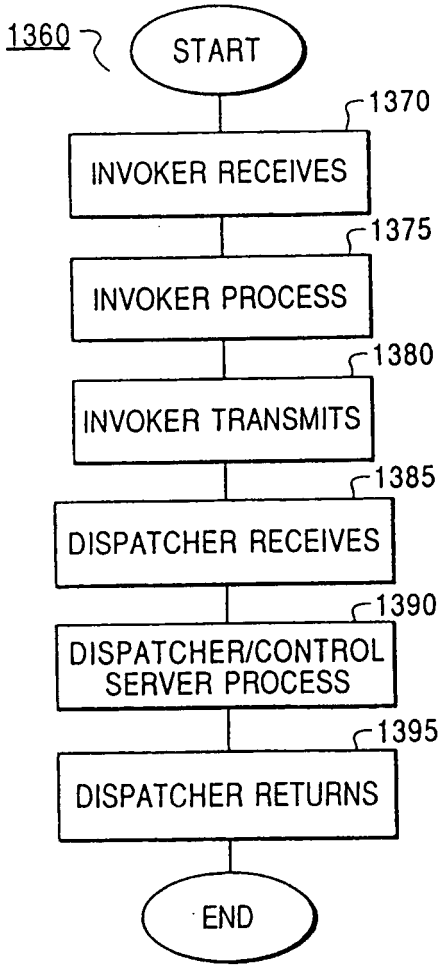

FIG. 13

EP 0 474 339 A2

METHOD OVERRIDE TABLE

1110

METHOD SELECTOR ATTRIBUTE NAME 1115	VALUE 1120
PLATFORM	VAX.VMS
INTERACTION - TYPE	BUILT_IN

FIG. 11B

SERVER NODE TABLE

1150

PLATFORM TYPE 1152	LOCATION 1154
TYPE A	NODE a, NODE b

FIG. 11C

CLASS DATABASE OVERRIDE TABLE

1170

DATABASE NAME 1172	LOCATION 1174
DB_SCH_LST	Db1, Db2

FIG. 11D

EP 0 474 339 A2

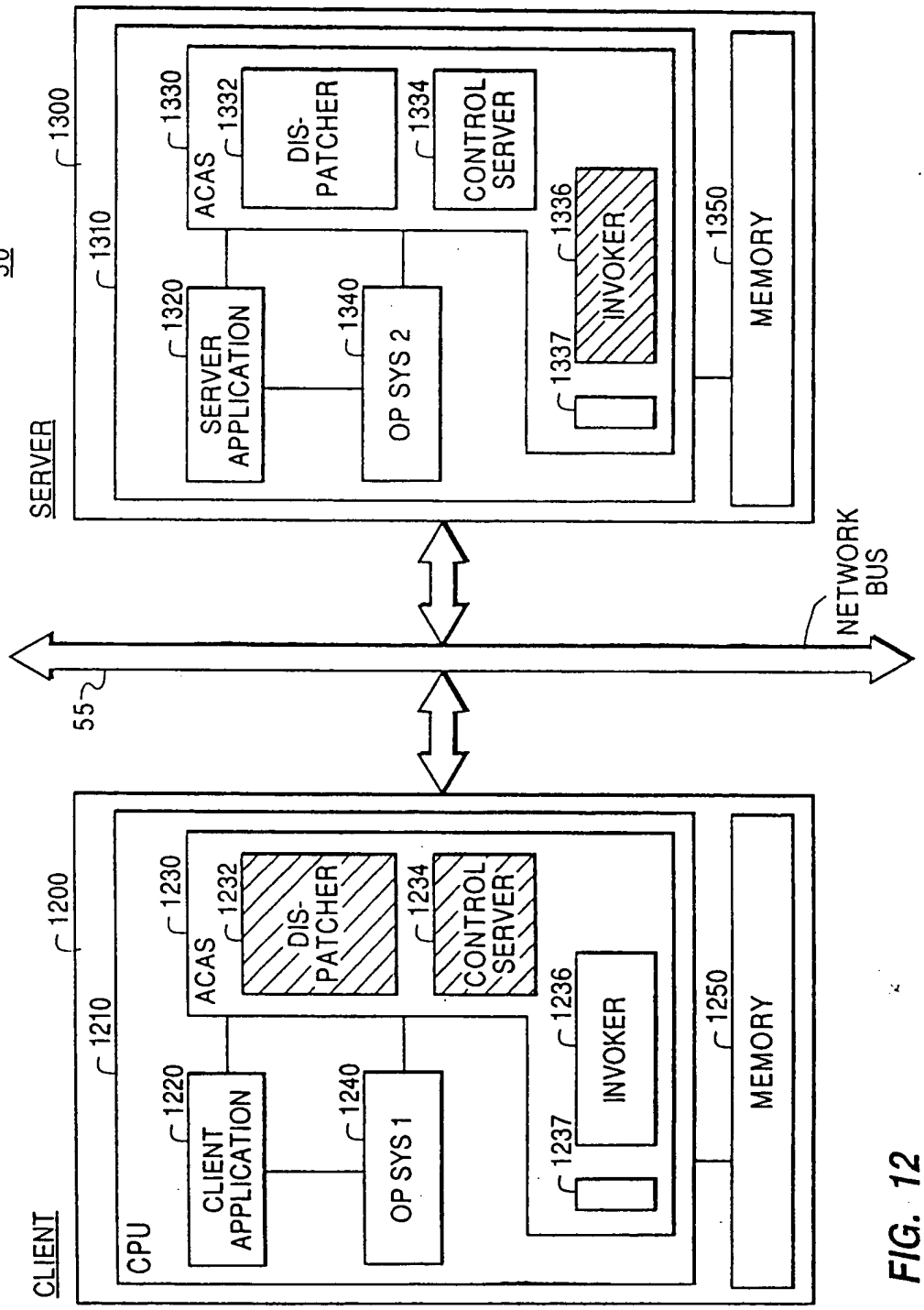
50
SERVER
1300
1310
1320
SERVER APPLICATION
1330
ACAS
1332
DIS-PATCHER
1334
CONTROL SERVER
1336
INVOKER
1337
1340
OP SYS 2
1350
MEMORY
NETWORK BUS
55
CLIENT
1200
1210
CPU
1220
CLIENT APPLICATION
1230
ACAS
1232
DIS-PATCHER
1234
CONTROL SERVER
1236
INVOKER
1237
1240
OP SYS 1
1250
MEMORY

FIG. 12

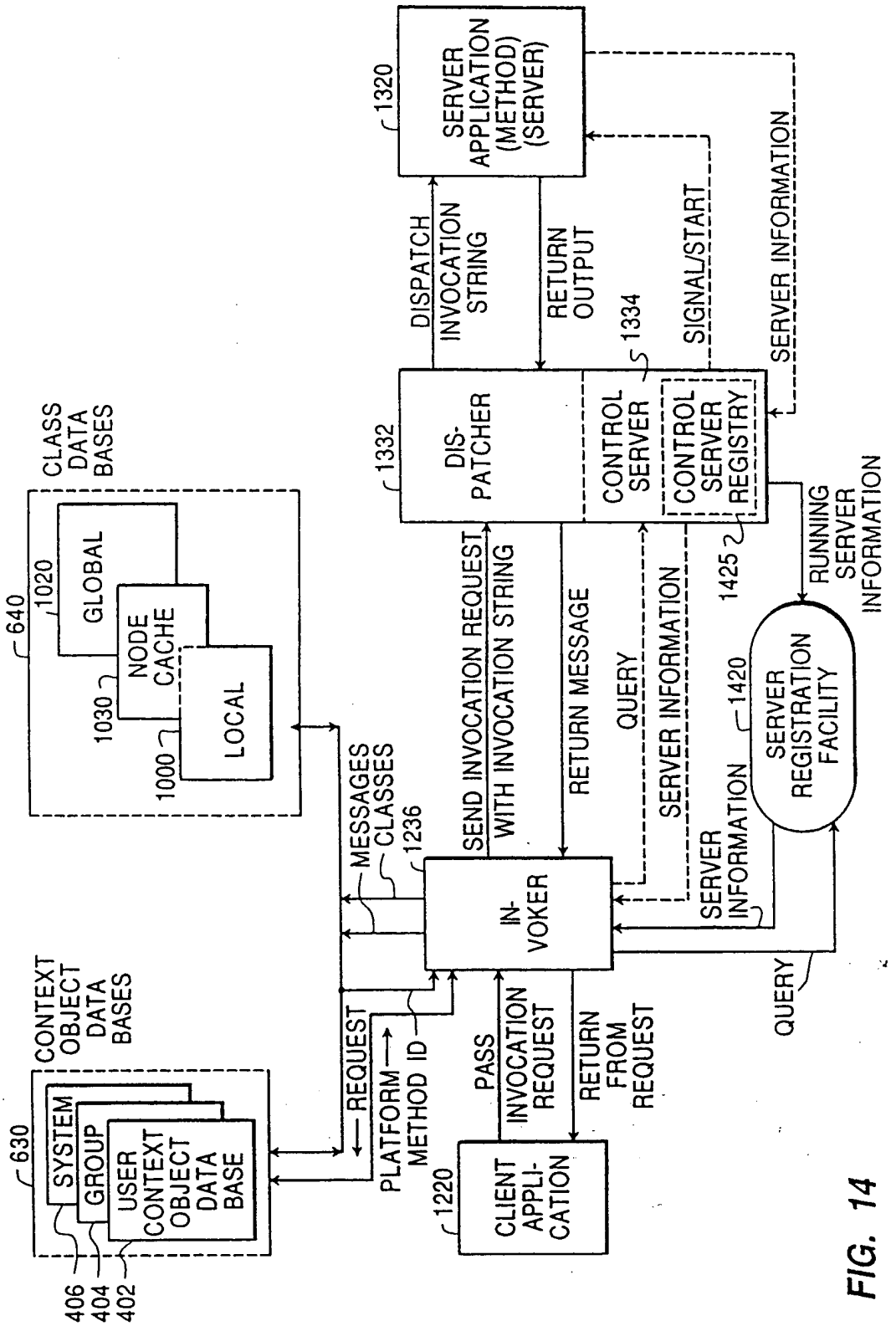
CLASS DATA BASES
640
1020
GLOBAL
NODE CACHE
1030
LOCAL
1000
CONTEXT OBJECT DATA BASES
630
SYSTEM
GROUP
USER CONTEXT OBJECT DATA BASE
1406
1404
1402
1320
SERVER APPLICATION (METHOD) (SERVER)
DISPATCH
INVOCATION STRING
RETURN OUTPUT
1334
SIGNAL/START
SERVER INFORMATION
1332
DIS-PATCHER
CONTROL SERVER
CONTROL SERVER REGISTRY
1425
RUNNING SERVER INFORMATION
SEND INVOCATION REQUEST
WITH INVOCATION STRING
RETURN MESSAGE
QUERY
SERVER INFORMATION
1420
SERVER REGISTRATION FACILITY
SERVER INFORMATION
MESSAGES CLASSES
1236
IN-VOKER
REQUEST
PLATFORM METHOD ID
PASS INVOCATION REQUEST
RETURN FROM REQUEST
1220
CLIENT APPLI-CATION
QUERY

FIG. 14

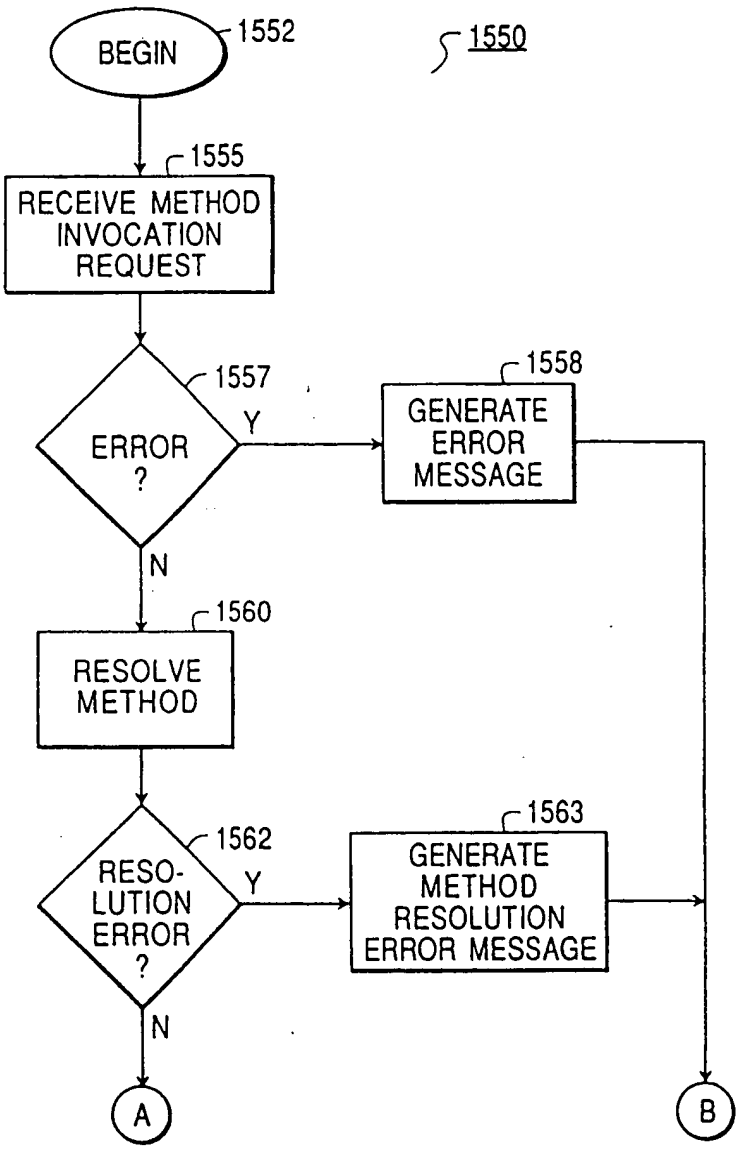

FIG. 15A
MAIN CONTROL PROCEDURE

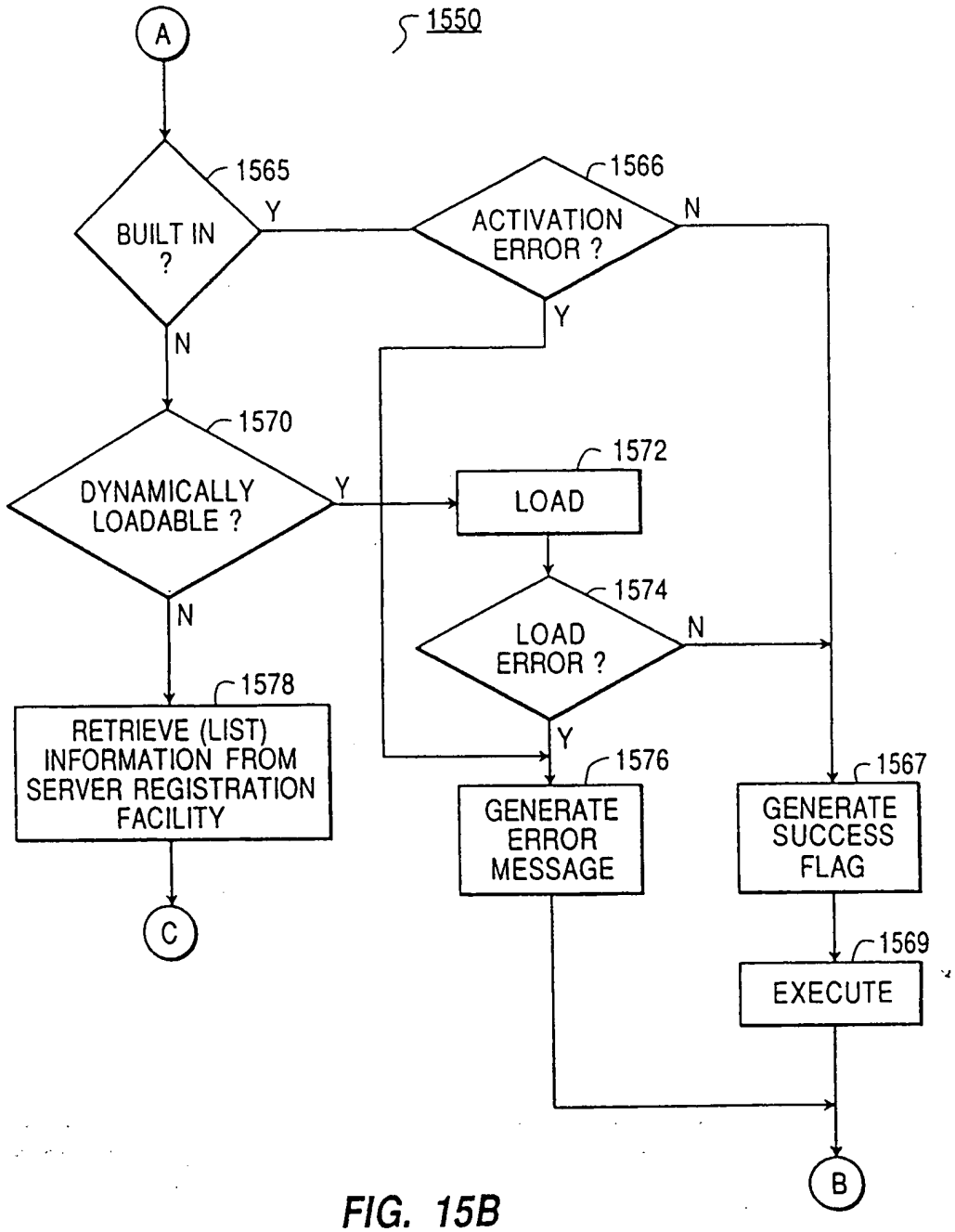
A
1550
1565
BUILT IN ?
Y
N
1566
ACTIVATION ERROR ?
N
Y
1570
DYNAMICALLY LOADABLE ?
Y
N
1572
LOAD
1574
LOAD ERROR ?
N
Y
1578
RETRIEVE (LIST) INFORMATION FROM SERVER REGISTRATION FACILITY
C
1576
GENERATE ERROR MESSAGE
1567
GENERATE SUCCESS FLAG
1569
EXECUTE
B

FIG. 15B

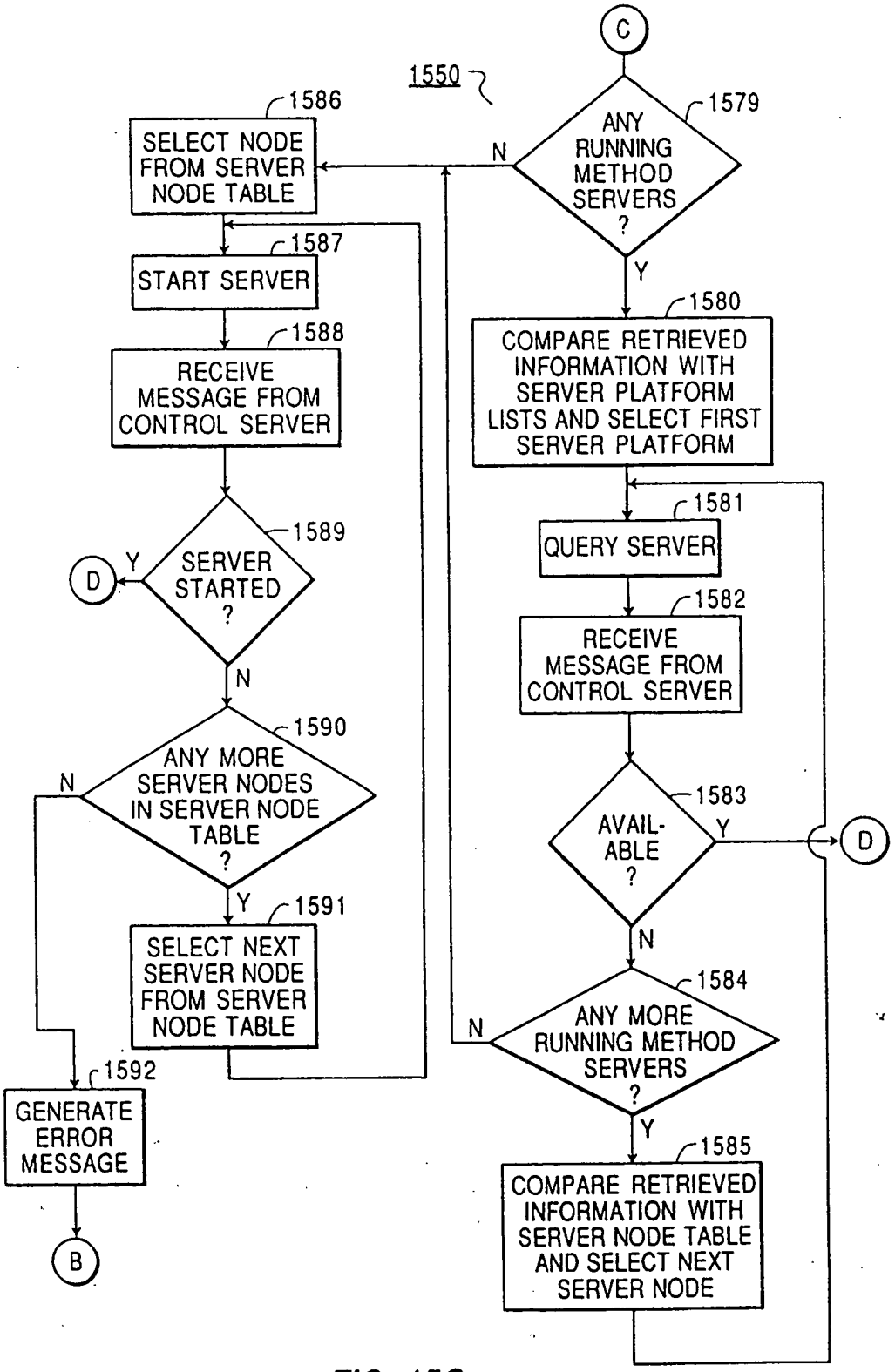
C
1550
1579
ANY RUNNING METHOD SERVERS ?
N
Y
1586
SELECT NODE FROM SERVER NODE TABLE
1580
COMPARE RETRIEVED INFORMATION WITH SERVER PLATFORM LISTS AND SELECT FIRST SERVER PLATFORM
1587
START SERVER
1581
QUERY SERVER
1588
RECEIVE MESSAGE FROM CONTROL SERVER
1582
RECEIVE MESSAGE FROM CONTROL SERVER
1589
SERVER STARTED ?
Y
D
N
1583
AVAIL-ABLE ?
Y
D
N
1590
ANY MORE SERVER NODES IN SERVER NODE TABLE ?
N
Y
1584
ANY MORE RUNNING METHOD SERVERS ?
N
Y
1591
SELECT NEXT SERVER NODE FROM SERVER NODE TABLE
1585
COMPARE RETRIEVED INFORMATION WITH SERVER NODE TABLE AND SELECT NEXT SERVER NODE
1592
GENERATE ERROR MESSAGE
B

FIG. 15C

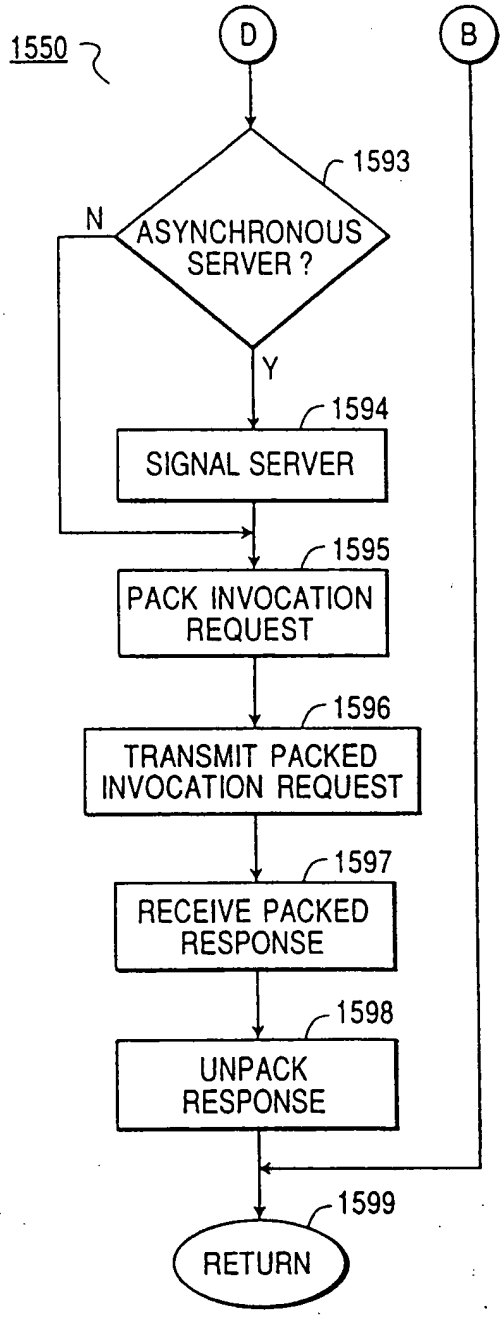
1550
D
B
1593
N
ASYNCHRONOUS SERVER ?
Y
1594
SIGNAL SERVER
1595
PACK INVOCATION REQUEST
1596
TRANSMIT PACKED INVOCATION REQUEST
1597
RECEIVE PACKED RESPONSE
1598
UNPACK RESPONSE
1599
RETURN

FIG. 15D

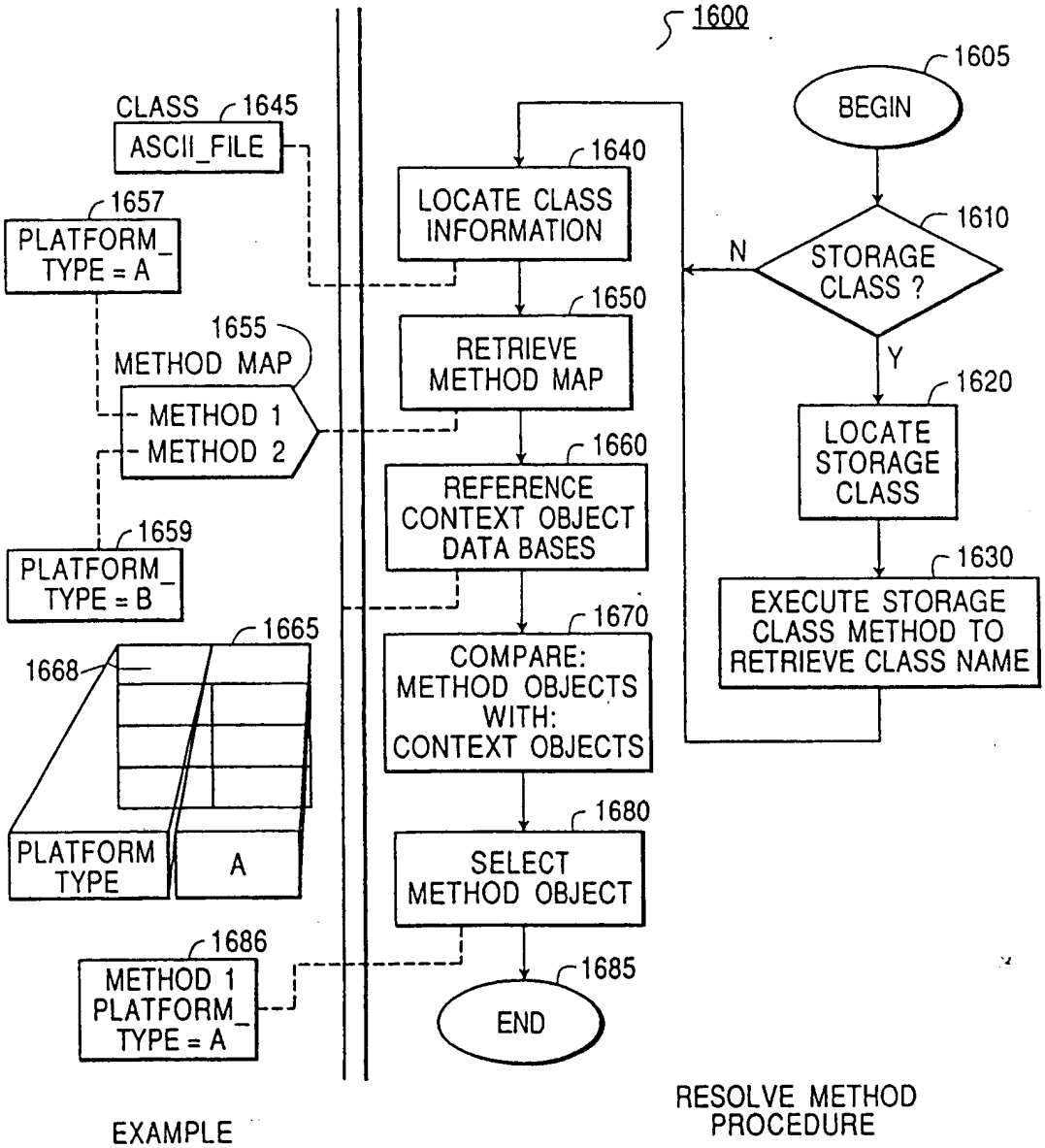
1600
1605
BEGIN
1610
STORAGE CLASS ?
N
Y
1620
LOCATE STORAGE CLASS
1630
EXECUTE STORAGE CLASS METHOD TO RETRIEVE CLASS NAME
RESOLVE METHOD PROCEDURE
1640
LOCATE CLASS INFORMATION
1650
RETRIEVE METHOD MAP
1660
REFERENCE CONTEXT OBJECT DATA BASES
1670
COMPARE: METHOD OBJECTS WITH: CONTEXT OBJECTS
1680
SELECT METHOD OBJECT
1685
END
CLASS
1645
ASCII_FILE
1657
PLATFORM_ TYPE = A
1655
METHOD MAP
METHOD 1
METHOD 2
1659
PLATFORM_ TYPE = B
1665
1668
PLATFORM TYPE
A
1686
METHOD 1 PLATFORM_ TYPE = A
EXAMPLE

FIG. 16

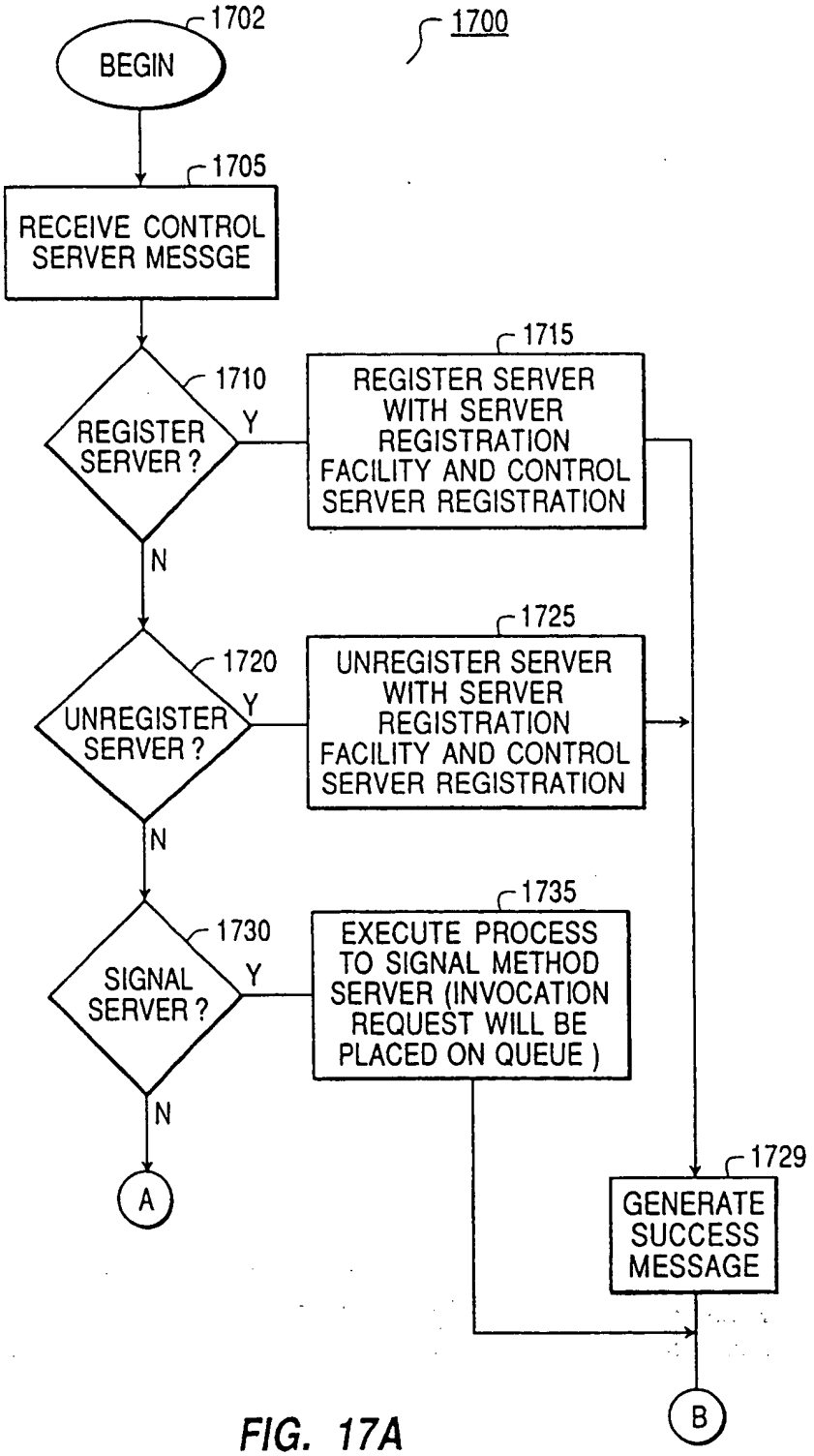

FIG. 17A
CONTROL SERVER

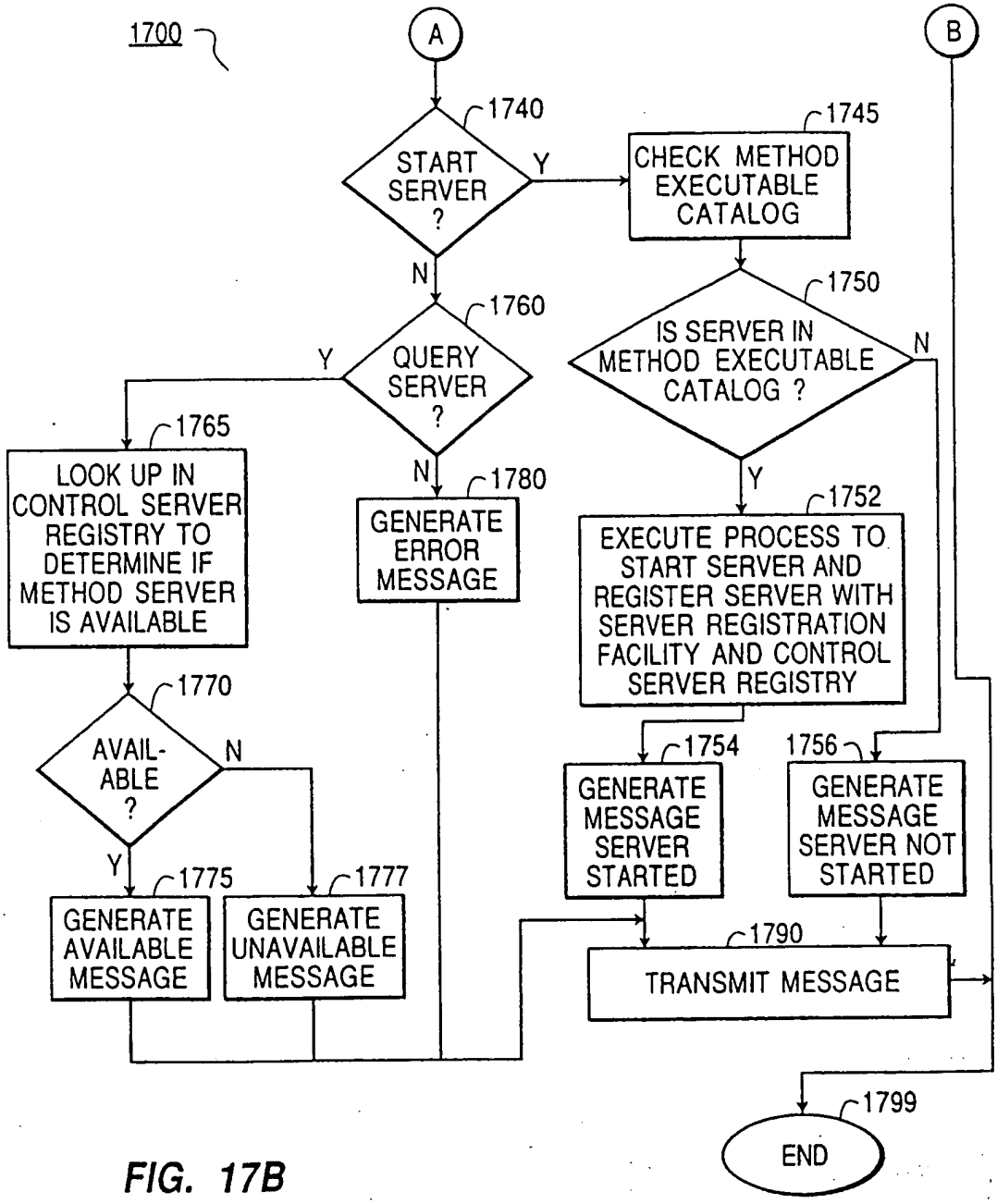

FIG. 17B
CONTROL SERVER

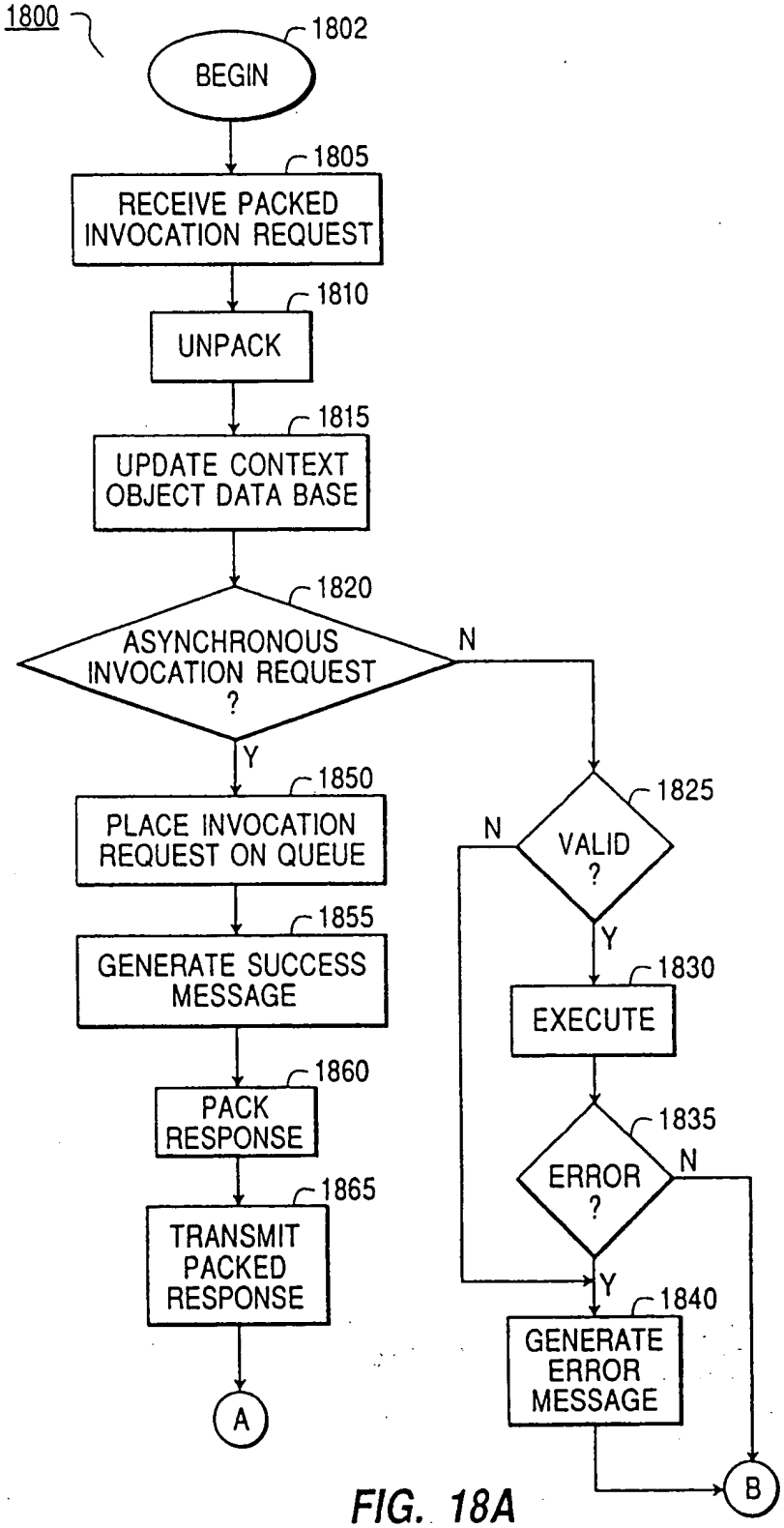

FIG. 18A
DISPATCHER PROCEDURE

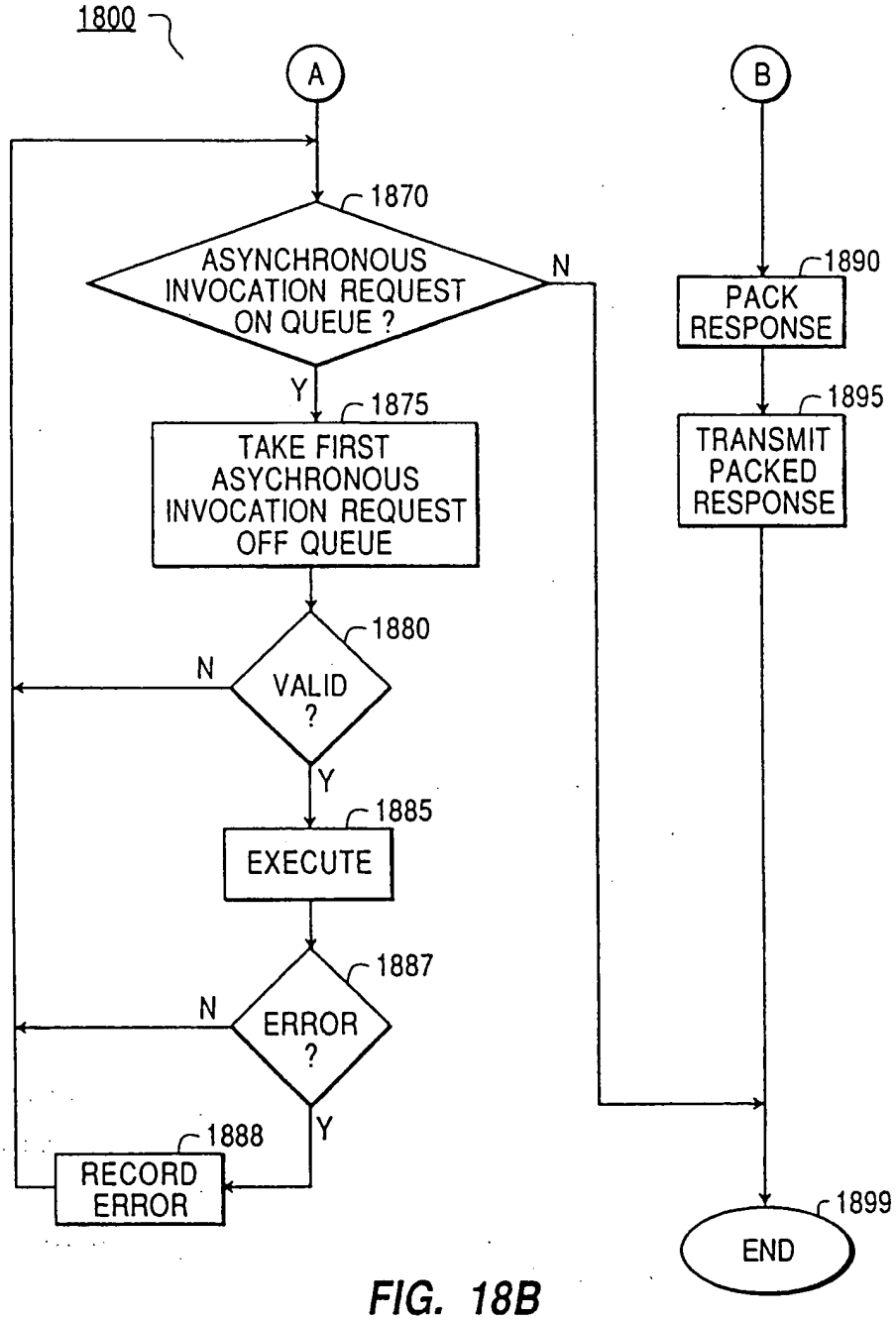

FIG. 18B
DISPATCHER PROCEDURE

Office européen des brevets

(11) Publication number: **0 474 339 A3**

(12) EUROPEAN PATENT APPLICATION

(21) Application number: **91306127.1**

(51) Int. Cl.5: **G06F 9/44**

(22) Date of filing: **05.07.91**

(30) Priority: **14.08.90 US 567303**

(43) Date of publication of application:
11.03.92 Bulletin 92/11

(84) Designated Contracting States:
DE FR GB IT NL

(88) Date of deferred publication of the search report:
13.01.93 Bulletin 93/02

(71) Applicant: **DIGITAL EQUIPMENT CORPORATION**
111 Powdermill Road
Maynard Massachusetts 01754-1418(US)

(72) Inventor: **Travis, Robert L., Jr.**
1547 Main Street
Concord, Massachusetts 01742(US)
Inventor: **Jacobson, Neal F.**
6 Cranwell Court
Nashua, New Hampshire 03062(US)
Inventor: **Wilson, Andrew P.**
Commons Brink, Bunces Lane, Burghfield Common
Reading RG7 3DP(GB)
Inventor: **Renzullo, Michael J.**
7 Byron Road
Ashland, Massachusetts 01721(US)

(74) Representative: **Goodman, Christopher et al**
Eric Potter & Clarkson St. Mary's Court St. Mary's Gate
Nottingham NG1 1LE(GB)

(54) Methods and apparatus for providing a client interface to an object-oriented invocation of an application.

(57) In response to a message requesting a method invocation from an application or user, a client application determines the proper method to be invoked by retrieving information from a class data base, comparing the retrieved information with user preferences, and selecting the proper method based upon the comparison. Server connection and start-up involves locating a platform capable of executing code associated with the selected method and, if necessary, executing a process to start an application associated with the selected method.

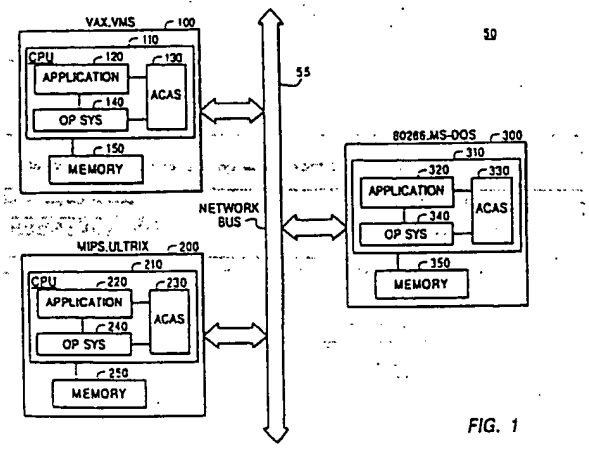

FIG. 1

European Patent Office

EUROPEAN SEARCH REPORT

Application Number

EP 91 30 6127

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int. Cl.5)
A	TECHNIQUE ET SCIENCE INFORMATIQUES vol. 6, no. 2, 1987, PARIS FR pages 166 - 169 M.SHAPIRO ET AL. 'SOS: un système d'exploitation réparti fondé sur les objets' * page 167, column 1, paragraph 2 - page 169, column 1, paragraph 2 *	1,19,39, 44	
P,A	J. FORGIONE ET AL. 'Response to the OMG RFI for the Object Request Broker' 15 August 1990 , DATA GENERAL CORPORATION , WESTBORO, MA, USA * the whole document *	1-45	
			TECHNICAL FIELDS SEARCHED (Int. Cl.5)

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	05 NOVEMBER 1992	TALLOEN J.M.F.S.

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P0401)